Universidad
Carlos III de Madrid

PROYECTO FIN DE CARRERA

Estudio Comparativo de las capacidades de Intel y ARM

Autor: ***Andrés Felipe Mesa Zúñiga***
Director: ***Luis Miguel Sánchez García***
Tutor: ***Rafael Sotomayor Fernández***

AGRADECIMIENTOS

Me gustaría agradecer a la Universidad Carlos III, y a todos sus profesores, su apoyo a lo largo de todos los años que he pasado en esta maravillosa Universidad, sin duda, los mejores de mi vida.

A mi familia, el apoyo y la paciencia que me brindan cada día en cada uno de los retos que afronto, sin ellos nunca hubiera llegado hasta aquí y sin ellos nunca llegaría a ningún lado. Sin el ánimo y el aliento que me ofrecen día tras día no podría haber escrito estas líneas.

A mis tutores Luís Miguel y Rafael, por hacerlo todo tan sencillo, por su brillantez a la hora de orientarme y la paciencia demostrada al ser asaltados con mis dudas. Han sido una inspiración a lo largo de la carrera y los considero unos profesionales excepcionales.

A mis amigos y compañeros, por su insuperable compañía y dedicación. Sin ellos no hubiera podido afrontar esta carrera ni hubiera disfrutado tanto de estos años.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En estos momentos los procesadores con arquitectura ARM se encuentran presentes en prácticamente todos los teléfonos móviles de nueva generación y en una innumerable lista de dispositivos electrónicos modernos. Su principal ventaja reside en su capacidad de procesamiento y su bajo consumo eléctrico, este tipo de procesadores RISC de nueva generación están destinados a evolucionar a la par que la tecnología misma. Otra de las grandes ventajas que aporta a la hora de ser usado es su baja emisión de calor, lo que los convierte en el perfecto aliado de la tecnología móvil.

La principal motivación de este proyecto es ahondar en la arquitectura ARM y comprobar, mediante el uso de la biblioteca de benchmarking Parsec 2.1, la verdadera capacidad que estos procesadores pueden ofrecer respecto a los que actualmente usamos para el procesamiento de grandes cargas de trabajo. En el caso que nos ocupa, la arquitectura que usaremos para comparar los datos es la ofrecida por los procesadores Intel x86.

La biblioteca Parsec 2.1 es a juicio del autor de este proyecto, una gran herramienta a la hora de cuantificar el rendimiento ofrecido por los procesadores Intel. La batería de pruebas está pensada para procesadores Intel y ha sido compilada para dicha arquitectura, nuestro trabajo tendrá como paso intermedio la compilación de dicha biblioteca para la arquitectura ARM así como todas las herramientas que emplea.

Una vez conseguido y establecido el entorno de pruebas procederemos a evaluar los resultados obtenidos por el procesador ARM y lo compararemos con los resultados obtenidos por Intel.

Cabe reseñar que este trabajo solo desempeña una función académica y que en ningún momento se generará ningún material con posibles usos comerciales.

Esta memoria pretende servir para:

1. Exponer claramente la motivación y los objetivos de este estudio.
2. Detallar el estado actual de todas las tecnologías involucradas.
3. Comparar posibles alternativas y justificar las decisiones adoptadas a la hora de realizar este estudio.
4. Ofrecer los resultados y evaluar si se han cumplido los objetivos del proyecto.
5. Obtener conclusiones fehacientes a raíz del trabajo realizado.
6. Detallar como se ha planificado y ejecutado el proyecto así como los costes asociados al mismo.
7. Considerar la aportación de este estudio al estado actual del arte.
8. Ofrecer posibles aplicaciones derivadas de este proyecto.

1.1 MOTIVACIÓN

La informática actual cada día demanda una mayor capacidad de procesamiento, mayor rendimiento y menor consumo de recursos. Estos factores, unidos a la tendencia imparable hacia las tecnologías móviles, implican un gran cambio en el desarrollo de procesadores de datos y en la informática en general.

El enfoque de los procesadores hacia una frecuencia de reloj cada vez más alta ha llegado a su fin, dadas las limitaciones físicas que impiden seguir aumentando los ciclos de reloj indefinidamente (1). La opción que se ha postulado con más fuerza y que actualmente lidera la fabricación de procesadores es la opción de aunar varios núcleos de procesamiento en una misma máquina. Este cambio supone nuevos retos tanto para la fabricación de los componentes como para la programación de aplicaciones que sean capaces de aprovechar esta nueva característica arquitectónica.

El principal reto que plantea el aumento de procesadores es el poder armonizar correctamente el uso en paralelo de varios núcleos de procesamiento. Es por tanto primordial el estudio de las implicaciones que supone dividir el trabajo entre varias unidades. Los problemas de sincronización se han convertido en habituales para desarrolladores software y arquitectos hardware. A medida que aumenta el número de núcleos este problema se escala. El control sobre muchísimas maquinas que a su vez contienen varios procesadores supone un esfuerzo extra.

Las soluciones desarrolladas por fabricantes como Intel o AMD han sido ampliamente adoptadas por el PC, servidores y grandes centros de procesamiento de datos. Las veremos a lo largo de este documento cuando describamos el estado del arte.

Paralelamente al desarrollo de los procesadores multinúcleo, la telefonía móvil ha adoptado progresivamente el modelo propuesto para los PC, convirtiéndose en participe de este cambio de paradigma.

Los teléfonos móviles han evolucionado progresivamente a pequeños ordenadores de bolsillo, enfrentándose igualmente a los retos que la computación multinúcleo plantea. (2)

Para el caso de la telefonía móvil, el fabricante predominante a lo largo de los últimos años ha sido ARM, presente primeramente en un alto porcentaje de dispositivos empotrados. El estándar de fabricación de ARM ha sido adoptado por prácticamente todos los fabricantes de terminales móviles actuales.

La aparición de este tipo de procesadores, que funcionan en un dispositivo tan pequeño como un teléfono móvil, o incluso un reloj, plantea muchas cuestiones que motivan este documento.

Uno de los aspectos menos cuidados a la hora de usar procesadores cada vez con más núcleos es el aspecto energético, está ampliamente aceptado que a mayor número de procesadores, mayor será la demanda energética. Además cabe mencionar la ley de Moore, que ha servido como

medidor de la tendencia a aumentar la integración de los transistores a una escala menor cada vez, deriva en un aumento de los componentes y por tanto una demanda energética que aumenta. (3)

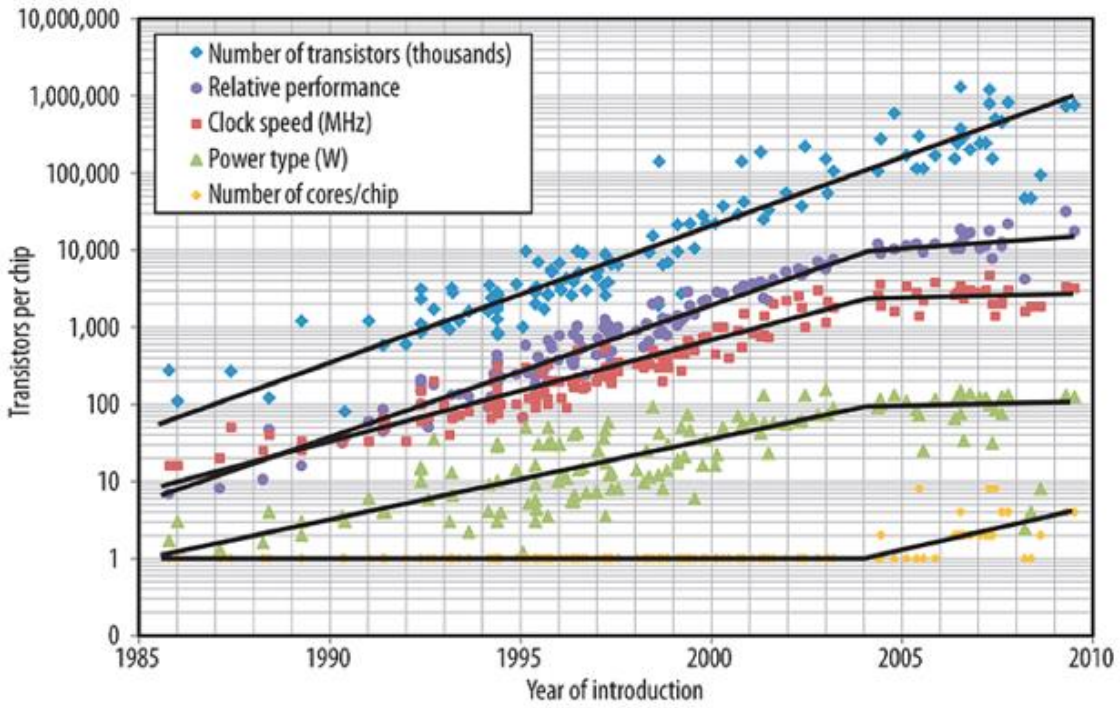

Figura 1 Aumento del nº de transistores, consumo de energía, nº de CPUs y rendimiento (Ley de Moore).

Este hecho es inaceptable en dispositivos móviles, puesto que el usuario espera poder usar su terminal sin tener que preocuparse por la batería, o al menos con un compromiso razonable entre rendimiento y consumo. Esto abre la puerta a la principal pregunta que trataremos de contestar en este trabajo:

¿Puede un procesador pensado para tecnología móvil, como los desarrollados por ARM competir con un procesador "convencional" en rendimiento y ahorro energético?

Esta cuestión es el pilar fundamental sobre el que versa esta investigación y por tanto trataremos de darle respuesta a lo largo de todo este documento. Esta pregunta obliga a cuestionarse si existe algún modo fiable de cuantificar el rendimiento de un procesador.

Necesitamos servirnos de algo fundamental a la hora de comprobar el rendimiento de un procesador, las herramientas de benchmarking. Existe una gran cantidad de suites de benchmarking implementadas para medir fiablemente el rendimiento que desempeña un procesador (4). Al ser sometido a condiciones reales en las cuales son puestos a máximo rendimiento, se puede evaluar el desempeño real que tienen. Esto plantea la segunda pregunta clave de este documento.

¿Existe alguna suite de benchmarking que pueda ser usada para comparar el rendimiento de los procesadores ARM y de los procesadores de uso comercial?

El objetivo de este trabajo es contestar a ambas cuestiones.

1.2 OBJETIVOS

El principal objetivo de este proyecto es conseguir averiguar si un procesador de la arquitectura ARM disponible, puede ser una alternativa real en términos de coste, procesamiento y eficiencia energética respecto a la arquitectura x86 y x64 de Intel. Para ello se plantean los siguientes sub-objetivos:

- Comparar la arquitectura x86 y x64 de Intel con ARM y tratar de decidir a priori cual presenta mejores cualidades en lo relativo a procesamiento y eficiencia energética.

- Portar a la arquitectura ARM un *benchmark* que a juicio del autor sirva para dar medidas fiables del rendimiento de las arquitecturas a comparar. Para ello trataremos de comparar que compiladores se pueden usar para realizar este proyecto y usar el más adecuado.

- Realizar los test que permita el benchmark en las dos arquitecturas comparadas y ofrecer los resultados.

- Realizar pruebas empleando diversos modos dentro del benchmark, pruebas secuenciales y en paralelo, con todas las técnicas y herramientas disponibles.

- Calcular el Speed-Up derivado de la aplicación de técnicas de paralelización.

- Calcular el coste del recompilado de la librería de benchmarking seleccionada.

- Estimar los costes de energía durante la ejecución del benchmarking.

- Decidir a posteriori si la arquitectura ARM supone algún tipo de mejora respecto a la arquitectura x86 y en qué puntos se considera válida.

- Tratar de comparar las pruebas con *frameworks* paralelos de trabajo con estudios previos de la universidad.

2. ESTADO DEL ARTE

En este apartado se define toda la tecnología involucrada en el proyecto. Para ello ofreceremos una visión global de cada una de las tecnologías involucradas en el del proyecto.

Dichas tecnologías son

- Arquitectura ARM
- Arquitectura X86 Y X64 INTEL
- *Frameworks* de paralelismo
  - Pthreads
  - TBB
  - OpenMP
- *Benchmarks*
  - Parsec 2.1
- Perf

2.1 ARQUITECTURA ARM

2.1.1 HISTORIA DE ARM

Figura 2 Logo ARM

En 1983 Acorn Computers Ltd comenzó el proyecto de desarrollar su propio modelo de procesador con un juego reducido de instrucciones (*RISC)*. Terminado el diseño preliminar los primeros prototipos del procesador en el año 1985, al que llamaron ARM1 (*Acorn RISC Machine 1).* El modelo contaba con apenas 25.000 transistores. Fue rápidamente sustituido por su versión mejorada ARM2 poco más de un año después, considerado el procesador *RISC* más simple del mundo (4). Poseía un bus de datos de 32 bits y ofrecía un espacio de direcciones de 26 bits, junto con 16 registros de 32 bits. Acorn había introducido con su segundo modelo un nuevo procesador, un nuevo sistema operativo, pero no un entorno de aplicaciones que ayudaran a los usuarios a usarlo masivamente. Esta situación llevo a dos años de duro trabajo para desarrollar aplicaciones nativas para la arquitectura.

Hasta la llegada de ARM3 en 1990 no se consiguieron latencias de 25Mhz. Este dato, unido al consumo reducido comparativamente con la arquitectura x86 consiguió que la empresa Apple, se fijara en ARM. Pretendían diseñar su primera PDA usando procesadores de baja potencia desarrollados por la empresa AT&T pero pronto quedaron fascinados por las ventajas que los

procesadores de Acorn podían ofrecer. Pese al escepticismo que suscitó esta unión y dada la poca madurez de la tecnología que ofrecía ARM la empresa Apple financió el continuo desarrollo de la empresa adquiriendo sus procesadores. La empresa continúo desarrollando sus procesadores y trató de continuar avanzando hacia el diseño de un procesador de propósito general. Las barreras económicas fueron grandes y afectaron al diseño de las siguientes generaciones de procesadores lo que limito ARM a una solución a escala baja. Únicamente la ilusión de sus diseñadores pudo superar esta barrera y finalizar el diseño del procesador ARM6 diseñado específicamente para Apple, mejorando notablemente el soporte para el controlador de video de sus *PDA's*. También consiguieron realizar grandes avances para dar soporte a los desarrolladores como ensambladores, compiladores, herramientas de test y hardware para evaluar antes de la salida de nuevos productos.

El núcleo mantuvo su simplicidad a pesar de los cambios: en efecto, el ARM2 tiene 30 000 transistores, mientras que el ARM6 sólo cuenta con 35 000. La idea era que el usuario final combinara el núcleo del ARM con un número opcional de periféricos integrados y otros elementos, pudiendo crear un procesador completo a la medida de sus necesidades .La mayor utilización de la tecnología ARM se alcanzó con el procesador ARM7TDMI.

La empresa DEC licenció el diseño, lo cual generó algo de confusión debido a que ya producía el DEC Alpha, y creó el *StrongARM*. Con una velocidad de reloj de 233 MHz, este procesador consumía solo 1 W de potencia (este consumo de energía se ha reducido en versiones más recientes). Esta tecnología pasó posteriormente a manos de Intel, como fruto de un acuerdo jurídico, que la integró en su línea de procesadores Intel i960 e hizo más ardua la competencia.

ARM consiguió atravesar grandes obstáculos gracias a sus socios, que seguían apostando por la tecnología desarrollada por Acorn pese al gran número de competidores que tenía y a día de hoy ha conseguido integrarse en el mercado más competitivo que existe, el de la telefonía móvil, con unos índices de venta de procesadores por encima de 10 mil millones de unidades (5).

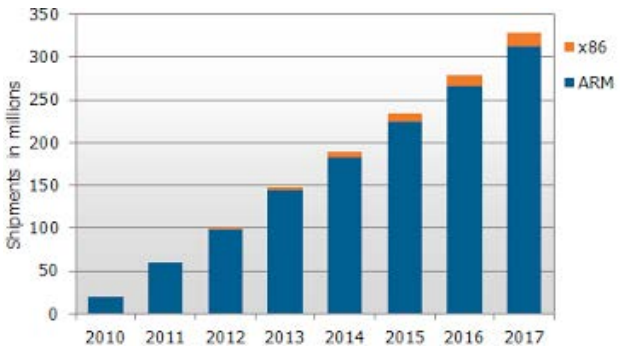

Figura 3 Expectativas de ventas de procesadores por fabricante. DisplaySearch, USA

Por tanto, la arquitectura ARM ha crecido hasta convertirse en la arquitectura más popular del planeta. Cerca del 75% de los procesadores de 32 bits poseen este chip en su núcleo. La arquitectura ARM ha sido utilizada en numerosos diseños y aplicaciones específicas para

productos estándar que pueden encontrarse actualmente en prácticamente todos los teléfonos móviles y la mayoría de los MP3, cámaras y sistemas de Navegación. También se utiliza en muchos productos de automoción, aplicaciones médicas e industriales.

Se han hecho intentos por llevar procesadores ARM al terreno de los servidores, como el proyecto desarrollado por Calxeda que pretendía establecer un servidor compuesto de 120 núcleos ARM Cortex-A9. Pese a que posteriormente se abandonó el proyecto, se pretendía conseguir aunar 480 núcleos en una sola máquina. Finalmente la compañía responsable perdió los fondos necesarios para continuar el proyecto.

La simplicidad de este tipo de procesadores los hace idóneos para un gasto energético muy bajo. El tamaño de la implementación, el rendimiento y el bajo consumo de energía son la base del desarrollo de este tipo de procesadores.

2.1.2 CARACTERISTICAS ACTUALES DE LA ARQUITECTURA ARM

Los procesadores ARM son un claro ejemplo de arquitectura RISC.

La arquitectura de 32-bits de ARM es la más utilizada actualmente en dispositivos móviles. En la actualidad la familia de procesadores Cortex en sus variantes A, R, M han sido empleados en la inmensa mayoría de dispositivos y se diferencian únicamente en el propósito para el cual han sido diseñados.

- **Cortex-A**: Destinados a aplicaciones.
- **Cortex-R**: Destinados a sistemas en tiempo real y empotrados.
- **Cortex-M**: Destinados a micro-controladores empotrados.

ARM es una máquina con juego reducido de instrucciones e incorpora las características de dichas maquinas (6):

- Un fichero de registro único.
- Una arquitectura de *load/store* donde las operaciones de procesamiento de datos solo operan sobre los contenidos de los registros, no directamente sobre la memoria.
- Simplicidad en los modos de direccionamiento. Las direcciones a *Load/store* están determinadas por el contenido de los registros.
- Cuenta con campos de instrucciones simplificadas y reducidas para simplificar la decodificación de instrucciones.

Además la arquitectura ARM ofrece:

- Control sobre la ALU y el decodificador de instrucciones a fin de maximizar su uso.
- Auto- incremento y decremento de modos de direccionamiento y optimización de bucles.

- Operaciones *load/store* múltiples para maximizar el *data throughput.*
- Ejecución de casi todas las instrucciones condicionales para maximizar el lanzamiento de ejecuciones.

Estas mejoras respecto a las máquinas RISC caracterizan a las maquinas ARM otorgándoles un buen balance entre rendimiento, pequeño tamaño de código, un bajo consumo eléctrico y una integración perfecta en una base de silicio.

En la siguiente ilustración podemos apreciar la distribución de un procesador ARM muy utilizado actualmente en una gran variedad de dispositivos móviles

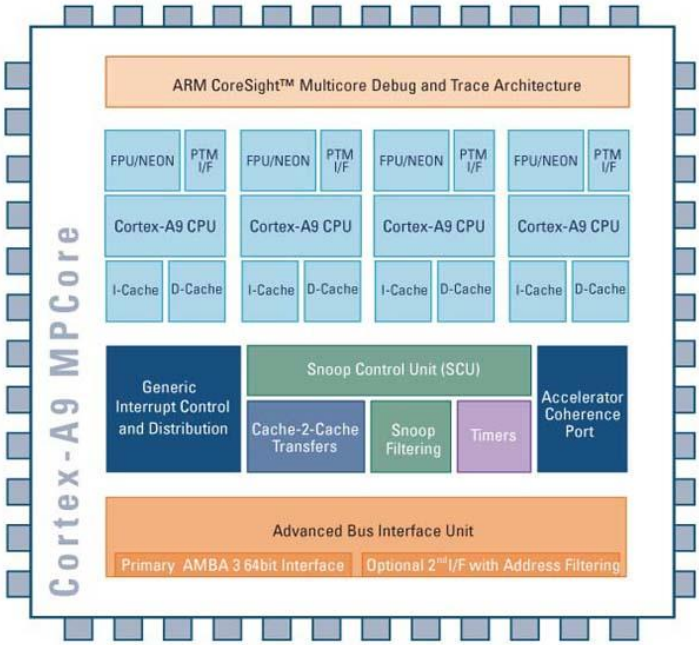

Figura 4 Diagrama de un Procesador ARM de 4 núcleos

En la imagen superior podemos ver plasmada la visión de componentes de un procesador ARM de última generación con sus 4 núcleos, sus cachés de procesamiento paralelo, su bus de datos, su controlador de interrupciones y de cachés.

MODOS DE CPU

Todos los diseños de ARM han sido diseñados para soportar diferentes modos de ejecución, exceptuando por motivos obvios a la familia M destinada a micro-controladores. Siguiendo las líneas generales seguidas por todos los fabricantes de núcleos, los procesadores únicamente pueden funcionar en uno de los siguientes modos de ejecución (6 pág. 41) :

No privilegiados

1. Usuario: No privilegiado, destinado a ejecución sin llamadas que requieran interrupciones o llamadas al sistema.

Privilegiados

2. FIQ: para interrupciones rápidas con baja latencia
3. IRQ: para el manejo normal de interrupciones software.
4. Supervisor: destinado al arranque y a las instrucciones de supervisor, operaciones destinadas para acceder a recursos del sistema operativo.
5. *Abort*: Destinado a manejar las excepciones producidas por la MMU, errores de chache de Nivel 1 y 2, errores en operaciones *Prefech* y errores de datos.
6. Indefinido: Para excepciones no definidas.
7. Sistema: Destinado a compartir los registros del modo usuario sin emplear una excepción y realizar operaciones privilegiadas.
8. Monitor: usado como modo seguro para pasar de un estado seguro a no seguro y viceversa del procesador.
9. *Hyp*: Aporta soporte para virtualización y operaciones no seguras.

REGISTROS EN ARM

Las máquinas ARM cuentan con 31 registros de propósito general, cada uno de 32 bits. 16 de ellos son visibles, los otros son usados para mejorar el procesamiento de excepciones. Todos los especificadores de registro pueden ser direccionados desde los 16 registros visibles. (6 pág. 43 a 53)

Lo 16 registros principales pueden ser usados por código no privilegiado siempre en modo usuario. Esto significa:

- En el modo usuario únicamente se puede cambiar a otro modo generando una excepción.
- El sistema de memoria y los coprocesadores únicamente pueden dar la capacidad al modo usuario para acceder a ciertas partes de la memoria, obviamente menos que en modo privilegiado.

Los 16 registros visibles que provee se componen de:

Puntero de pila	Usado para las operaciones *PUSH* y *POP*
Registro de enlace	Este registro mantiene la dirección de la siguiente instrucción que se debe realizar tras cada *Branch and Link.* También se usa para devolver la información de dirección cuando se entra en modo excepción
Contador de programa	Usado para tareas tales como saber que instrucción es la que se está ejecutando en un determinado momento o saber si una instrucción es múltiple antes de ejecutarla.

Los 13 registros restantes no tienen ningún propósito para el hardware. Están definidos enteramente por el software ejecutado en ese momento. Por tanto son definidos por el usuario.

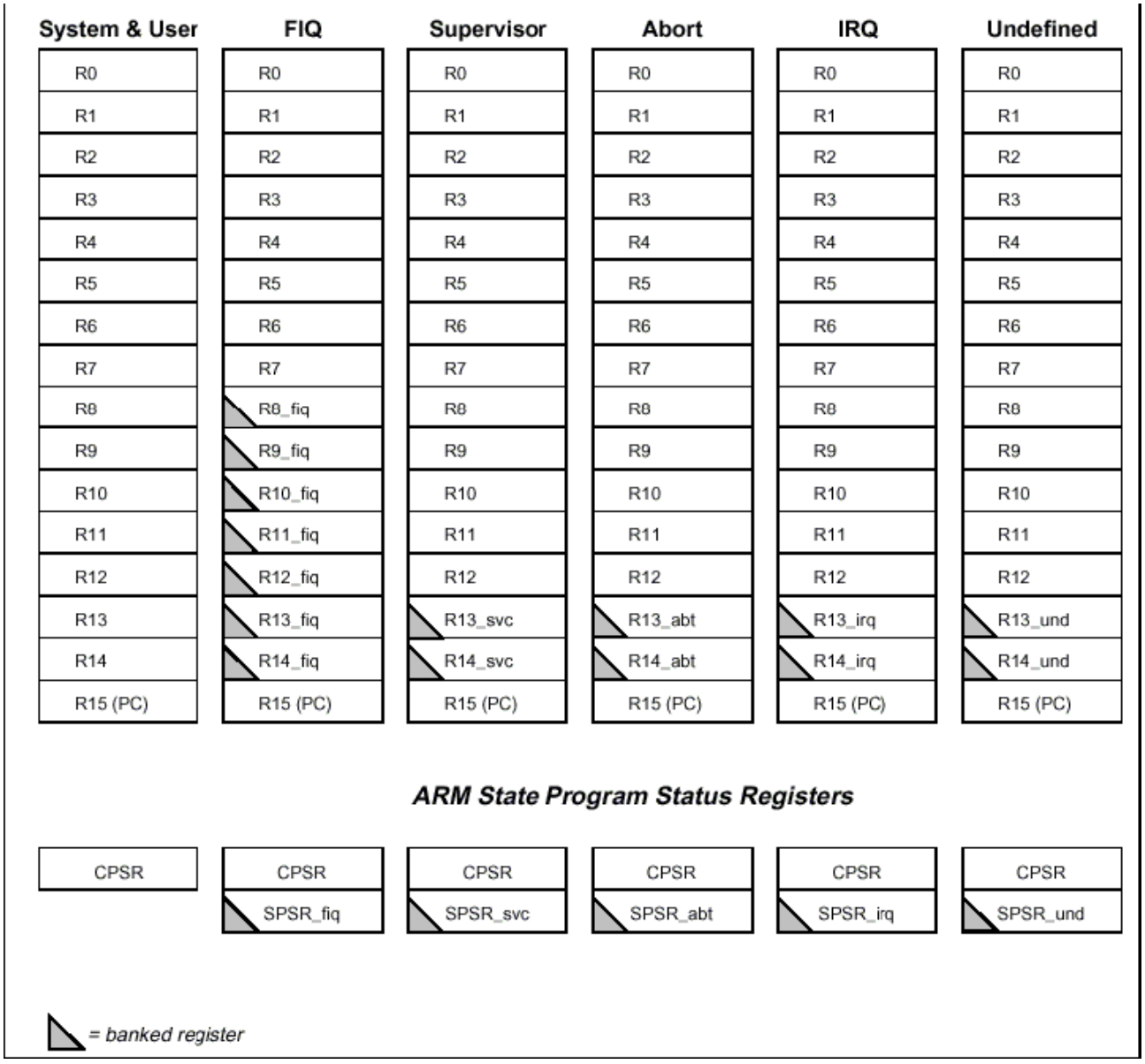

Figura 5 Relación de registros y modos en ARM

REGISTROS DE ESTADO

Todo el estado del procesador al margen de los registros de propósito general está contenido en los registros de estado. El estado actual en el que se encuentra el procesador se guarda en el registro *CPSR (Current Program Status Register*) que guarda:

- Cuatro *flags* de condición Negativa, Cero, Acarreo y Desbordamiento.
- Un *flag sticky.* Dicho *flag* codifica cuando se ha producido saturación en una instrucción aritmética o señala el desbordamiento producido en algunas operaciones de multiplicación acumulativas.
- Cuatro *flags* de Mayor o Igual que codifican a su vez los siguientes condiciones en instrucciones paralelas:
    - Cuando el resultado en operaciones con signo es no-negativo.
    - Cuando las operaciones sin signo producen acarreo o decremento de una unidad superior (en el caso de las restas).
- Dos bits de tipo de interrupción.
- Un bit para que en caso de error se pueda abortar una operación de modo síncrono o asíncrono.
- Cinco bits que codifican el modo actual en el que se encuentra el procesador actualmente.
- Dos bits para distinguir cualquier instrucción en ARM que están en ejecución.
- Un bit de control sobre la finalización de operaciones *Load/Store.*

Cada excepción cuenta además con un SPSR (*Saved Program Status Register)* que mantiene el CPSR de la tarea en el momento exacto anterior al lanzamiento de la excepción. Ambos registros se acceden con instrucciones especiales.

EL JUEGO DE INSTRUCCIONES EN ARM

En este apartado daremos una descripción general de ellas y una visión general de cada tipo.

(6 pág. 109)

La arquitectura ARM de 32 bits, como ya hemos especificado, cuenta con un juego de instrucciones reducido o RISC. Por tanto incorpora las siguientes características RISC:

- Solo admite acceder a la memoria mediante operaciones LOAD y STORE
- No soporta Accesos a memoria no alineados, a excepción de ciertos núcleos de la familia M.
- Matriz de registros uniforme. 16x32 bits.
- Longitud de instrucciones de 32 bits arregladas para favorecer su decodificación y asignación a procesos de Pipeline. De este modo se trata de reducir la densidad de código.
- Trata de realizar únicamente una instrucción por ciclo de CPU.
- Ejecución condicional de muchas instrucciones para reducir el tiempo de los saltos, de este

modo trata de compensar la falta de predictor de salto.
- Las Instrucciones aritméticas alteran el registro de estado.
- Soporte para Instrucciones aritméticas para cálculo de direcciones sin pérdida de rendimiento.
- Diversos modos de direccionamiento optimizados.
- Registro que mantiene la dirección a la que volver tras realizar una función o llamada.
- Dos niveles de interrupción.

En ARM el juego de instrucciones se puede dividir en seis clases:

- Instrucciones de salto
- De procesamiento de datos
- De transferencia de registro de estado
- Carga y Almacenamiento
- Instrucciones de coprocesador
- Excepciones

La mayor parte de las instrucciones de procesamiento de datos y las instrucciones del coprocesador pueden actualizar los *flags* de condición del CPSR acorde a su resultado.

Casi todas las instrucciones en ARM contienen un campo 4 bits de condición. El valor uno especifica si la instrucción se ejecutara incondicionalmente.

Otros catorce valores especifican ejecución condicional de una instrucción. Si el *flag* de condición indica que la instrucción es verdadera en el momento que la instrucción comienza a ejecutarse entonces se ejecutara normalmente. En otro caso la instrucción no hará nada. Los catorce valores permiten:

- Verificar igualdad y desigualdad
- Verificar menor, menor-igual, mayor y mayor igual. Aplicable a ambas aritméticas, con y sin signo.
- Cada *flag* de estado que se debe comprobar individualmente.

El dieciseisavo valor del campo de estado codifica instrucciones alternativas. Nunca permite ejecución condicional.

El juego de instrucciones del ARM incluye características adicionales que le permiten conseguir un mejor rendimiento en su ejecución. Para mantener el concepto tradicional de RISC, se estableció la ejecución de una orden en un tiempo, por lo general, de un ciclo. La característica más interesante es el uso de los 4 bits superiores como código de condición, haciendo que cualquier instrucción pueda ser condicional. Este corte reduce el espacio para algunos desplazamientos en el acceso a la memoria, pero permite evitar perder ciclos de reloj en el pipeline al ejecutar pequeños trozos de código con ejecución condicional.

Dada la amplitud del juego completo de instrucciones, solo hemos presentado un breve resumen del juego de instrucciones. Se insta a consultar el manual completo de ARM indicado en las referencias.

INSTRUCCIONES DE PROCESAMIENTO DE DATOS

Realizan los cálculos sobre los registros de propósito general.

Hay cinco tipos de instrucciones:

Aritmético Lógicas	Comparten signatura, realizan una operación sobre dos operandos y escriben el resultado en un registro de destino.
De comparación de registros	Funcionan exactamente igual que las aritmético lógicas salvo que no escriben el resultado. En cambio actualizan los *flags* de condición.
Una instrucción, múltiples datos	Suman o restan cada operando en como dos operaciones paralelas de 16 bits, o 4 de 8 bits. Pueden ser con o sin signo.
Multiplicación	Son muy variadas y pueden ser consultadas en el anexo 1.
Operaciones adicionales	Operaciones para contar los primeros ceros de un registro, sumas de diferencias absolutas y acumulativas.

INSTRUCCIONES DE TRANSFERENCIA DE REGISTRO DE ESTADO

Transfieren el contenido de CPSR y SPSR o desde un registro de propósito general al CPSR. Pueden:

- Establecer los valores de los *flags* de estado.
- Establecer los valores de los bits de interrupción
- Alternar el modo en el que se encuentra el procesador
- Alterar el control de finalización en operaciones de Carga y Almacenamiento

INSTRUCCIONES DE SALTO

Con el fin de permitir muchas operaciones de procesamiento de datos que pueden incluir el cambio de flujo de la ejecución ARM incluye las operaciones de salto. Estas operaciones permiten un desplazamiento de 24 bits con signo para poder realizar saltos de hasta 32 Mb.

AMR provee además un tipo de salto distinto, el *salto y enlace*, que permite preservar la dirección de la instrucción después del salto en el registro 14, destinado para este tipo de saltos. Con una llamada al sistema se puede volver a saltar a dicha instrucción sobrescribiendo el contador de programa.

INSTRUCCIONES DE CARGA Y ALMACENAMIENTO DE REGISTROS

Las operaciones disponibles para carga y almacenamiento permiten guardar uno o varios registros. Además permiten hacerlo de modo exclusivo, sin interrupciones. Dichas operaciones son:

Carga y Almacenamiento de un registro	Pueden cargar o guardar 64, 32 y 16 bits u 8 bytes para la carga de un registro. Tienen 3 modos de direccionamiento, mediante offset, pre-indexado y post-indexado. Dado que el contador de programa es un registro de propósito general, puede ser usado para realizar un salto a cada una de las de las direcciones de memoria en un total de 4 GB.
Carga y Almacenamiento de múltiples registros	Usadas para realizar operaciones en bloque para cargar o guardar múltiples registros desde o hacia la memoria. Cuenta con cuatro modos de direccionamiento, pre y post incremento, pre y pos decremento.
Carga y Almacenamiento de un registro en modo exclusivo	Soportan sincronización a la hora de cargar y guardar un registro. Proveen soporte para la realización de operaciones atómicas y el uso de semáforos.

COPROCESADORES

ARM ofrece una forma no intrusiva de ampliar el juego de instrucciones mediante el uso de coprocesadores. El espacio del coprocesador está dividido lógicamente en 16 coprocesadores. Se reservan para funciones de control cómo manejar las caches y gestionar la memoria en algunas operaciones. Los periféricos habitualmente se asignan a uno de los coprocesadores dado que su latencia es más baja.

INSTRUCCIONES AL COPROCESADOR

El coprocesador puede realizar las siguientes instrucciones:

- Procesamiento de Datos
- Transferencia de Datos
- Transferencia de Registros

INSTRUCCIONES GENERADORAS DE EXCEPCIONES

El juego de instrucciones de las maquinas ARM incluye dos tipos de interrupciones generadas por software:

Interrupciones	Usadas para realizar llamadas al sistema para requerir un servicio del sistema operativo, cambian el modo del procesador a modo privilegiado. De este modo una tarea obtiene acceso a recursos y operaciones privilegiadas.
Breakpoints	Se producen una excepción y son útiles a la hora de depurar la programación de la máquina.

Adicionalmente se pueden añadir las excepciones no definidas producidas por:

- Instrucciones del coprocesador no reconocidas por el coprocesador hardware.
- Instrucciones no reconocidas

EXCEPCIONES

ARM soporta siete tipos de instrucciones, y para cada una de ellas cuenta con un modo privilegiado de procesamiento asociado. Estas son:

- *Reset*
- Ejecución de una instrucción no definida
- Interrupciones Software y llamadas al sistema
- *Prefetch Abort.*
- Abortado acceso a memoria
- IRQ, interrupción normal
- FIQ, Interrupción rápida.

Cuando una excepción ocurre, los registros son reemplazados por registros específicos del modo excepción. Todos los modos de excepción tienen registros de reemplazo para R13 y R14 y las

interrupciones rápidas tienen registros adicionales para el procesamiento rápido de las mismas.

Cuando el manejador de instrucciones es llamado, el registro 14 es usado para obtener la dirección en la que se encuentra el procesamiento de dicha excepción. Esto se usa para poder volver a la instrucción causante de la excepción una vez que esta es controlada.

EL registro 13 está guardado para todos los modos de excepción para proveer manejadores que tengan un puntero de pila privado. En el caso de la interrupción rápida, se guarda además el registro 8 y 12 con el fin de poder volver de la interrupción sin salvar dichos registros.

El modo de interrupción de Sistema usa los registros del modo Usuario. Es usado para tareas que requieren acceso privilegiado a memoria y los coprocesadores, no tiene limitaciones respecto a las excepciones que se puedan producir. Ocurre lo mismo que con las interrupciones Software.

EL PROCESO DE EXCEPCIÓN

Cuando una excepción ocurre, el procesador ARM detiene la ejecución en alguno de los modos definidos en el apartado anterior. Para ello emplea un arreglo de direccionamiento conocido como *vector de excepciones.* Hay un vector definido para cada tipo de excepción, incluyendo el *reset.*

El sistema operativo instalado ofrece para cada excepción un manejador adecuado. Las operaciones privilegiadas que normalmente son ejecutas en modo Sistema permiten que cuando una excepción salte el sistema no pierda su estado.

AMPLIACIONES DEL JUEGO DE INSTRUCCIONES

Para complementar el juego de instrucciones se cuenta con ampliaciones y diversas técnicas para ampliar la densidad del código compilado, a continuación describimos las principales ampliaciones que envuelven al juego principal.

THUMB

De 16 bits (2 bytes) de longitud por instrucción, en lugar de 32 bits (4 bytes) como el juego estándar de ARM. Es un subconjunto formado por las instrucciones que se usan con más frecuencia. El rendimiento es superior a un código de 32 bits en donde el puerto de memoria o ancho del bus de comunicaciones son menores a 32 bits.

THUMB-2

Extiende el soporte *Thumb* permitiendo manipulación a nivel de bit. Se recomienda de nuevo consultar la bibliografía para ampliar información.

JAZELLE

Permite que ciertos tipos de procesadores ejecuten Java *bytecode* nativamente en el hardware. En las versiones más modernas de los procesadores no se da soporte para la aceleración por hardware.

ARM Y EL SOPORTE DE 64 BITS

La introducción de la nueva arquitectura llamada AArch64 y el nuevo juego de instrucciones proporciona soporte para operaciones sobre registros de 64 bits. Proporcionando retro compatibilidad con el antiguo juego de instrucciones y las tecnologías de ampliación del juego de instrucciones.

Cuenta con las siguientes mejoras respecto a la arquitectura de 32 bits:

- 31 registros de propósito general.
- Registro de puntero de pila.
- Se deshabilita el acceso como registro al contador de programa.
- El direccionamiento se presupone siempre de 32 bits.
- Los argumentos pueden ser tanto de 32 como de 64 bits.
- Soporte para coma flotante de doble precisión.
- Mejora considerable del sistema de excepciones con menos registros y modos de procesador.

2.1.3 RELEVANCIA PARA EL PROYECTO DE LA ARQUITECTURA ARM

La arquitectura ARM supone en primer lugar un claro ejemplo de arquitectura RISC, que en contraposición a las arquitecturas x86 y x86-x64 de Intel emplea una lógica más sencilla para la ejecución de las aplicaciones. La implicaciones a nivel energético son amplias, puesto que un juego reducido de requiere de mayores instrucciones para realizar operaciones complejas. Se quiere comprobar el impacto que tiene en el rendimiento el uso de este tipo de arquitecturas y el uso que realiza de la memoria puesto que solo accede a ella con dos operaciones, de carga y almacenamiento.

Por otro lado, su uso se ha masificado en los últimos años y los estudios realizados sobre esta arquitectura son escasos, especialmente se ha encontrado poca documentación referente al apartado energético.

2.2 ARQUITECTÚRA X86 Y X64 INTEL

2.2.1 HISTORIA DE INTEL

Figura 6 Logotipo de Intel

Intel es el mayor fabricante de circuitos integrados del mundo, creadores de la arquitectura x86, fundada en 1968 no fue hasta 1971 cuando Intel presento su primer microprocesador: el Intel 4004. El Intel 4004 (i4004), un CPU de 4bits, fue el primer microprocesador en un sólo chip, así como el primero disponible comercialmente. Con el Intel 4004 se conseguía situar en placas de 0,25 centímetros cuadrados un circuito integrado que contenía 2300 transistores. El objetivo era reunir en un microprocesador todos los elementos necesarios para crear un ordenador, a excepción de los dispositivos de entrada y salida (teclado, pantalla, impresora, etc.) imposibles de miniaturizar.

En 1972, Intel anunció una versión mejorada de su procesador anterior. Era el 8008, y su principal frente a otros modelos, fue poder acceder a más memoria y procesar 8 bits. La velocidad de su reloj alcanzaba los 740KHz. Fue el primer microprocesador de 8 bits, implantado con tecnología PMOS, contaba con 48 instrucciones, podía ejecutar 300.000 operaciones por segundo y direccionaba 16 *Kbytes* de memoria. Su evolución fue el 8080 con una velocidad de reloj que alcanzaba los 2 *Mhz*. Al año siguiente, aparece en el mercado el primer ordenador personal, de nombre Altair, basado en la micro-arquitectura del Intel 8080. El procesador de este computador suponía multiplicar por 10 el rendimiento del anterior, gracias a sus 2 *Mhz* de velocidad. Este microprocesador también direccionaba 8 bits, tenía 78 instrucciones, su velocidad de operaciones era 10 veces mayor que la del 8008 y podía direccionar hasta 64 *Kbytes* de memoria. (7).

En junio de 1978 y 1979 hacen su aparición los microprocesadores 8086 y 8088 que pasaron a formar el IBM PC, equipo que salió al mercado en 1981. Los i8086 e i8088 se basaron en el diseño del Intel 8080 y el Intel 8085, y de hecho son compatibles a nivel de ensamblador con el i8080. Ambos tienen cuatro registros generales de 16 bits, que también pueden ser accedidos como ocho registros de 8 bits, con cuatro registros.

El 1 de Febrero de 1982, Intel daba un nuevo vuelco a la industria con la aparición de los primeros 80286 (el famoso ordenador"286") con una velocidad entre 6 y 25 *Mhz* y un diseño mucho más cercano a los actuales microprocesadores. El 286 tiene el honor de ser el primer microprocesador usado para crear ordenadores clones en masa. Gracias al sistema de "licencias cruzadas", aparece en el mercado el primer fabricante de clónicos "IBM compatible".

En 1985 Intel lanza el i80386, con arquitectura de x86. Fue empleado como la unidad central de proceso de muchos computadores personales desde mediados de los años 80 hasta principios de los 90. También conocido como 386, con una velocidad de reloj entre 16 y 40 *Mhz*. Este producto se destacó principalmente por ser un microprocesador con arquitectura de 32 bits. Posteriormente se desarrollaron varias mejoras que fueron plasmadas en los microprocesadores i80386 y 80386sx, sacrificaban el bus de datos para dejarlo en uno de 16 bits, pero a menor costo. Estos

procesadores irrumpieron con la explosión del entorno gráfico Windows, desarrollado por Microsoft unos años antes, pero que aún no había tenido la suficiente aceptación por parte de los usuarios. Hasta la aparición del Pentium, los grandes avances de Intel fueron la inclusión de la caché de nivel 1 integrada en el propio chip y una continuo aumento de la frecuencia. Pequeñas mejoras y optimizaciones una unidad de coma flotante y un caché unificado integrados en el propio circuito integrado del microprocesador y una unidad de interfaz de bus mejorada duplicando la capacidad del 386.

La gran revolución llega en 1993 partían de una velocidad inicial de 60 MHz, llegando a los 200 MHz, algo que nadie había sido capaz de augurar unos años antes. Con una arquitectura real de 32 bits, se usaba de nuevo la tecnología de .8 micras, con lo que se lograba realizar más unidades en menos espacio. Poseía un bus de datos. El Pentium poseía una arquitectura capaz de ejecutar dos operaciones a la vez, gracias a sus dos pipeline de datos de 32 bits cada uno, uno equivalente al i486DX (u) y el otro equivalente al 486SX (u). Poseía un bus de datos de 64 bits, permitiendo un acceso de memoria de 64 bits.

Toda la familia Pentium fue integrada masivamente en los ordenadores personales durante toda la década de 1990, durante los primeros años del siglo XXI se trató de reducir el tamaño de los procesadores y disminuir su consumo energético.

Principalmente hasta la llegada del procesador Pentium D en 2005 Intel no dio el salto definitivo al nuevo paradigma de varios núcleos. La idea era colocar 2 procesadores Pentium 4 metidos en un solo encapsulado (2 núcleos Prescott para el *core* Smithfield y 2 núcleos Cedar Mill para el *core* Presler). Su proceso de fabricación fue inicialmente de 90 *nm* y en su segunda generación de 65 *nm*. En 2006 Intel anuncia la nueva generación: Xeon Dual Core con tecnología de doble núcleo.

Este nuevo procesador brindaba un 80% más de rendimiento por vatio y en un 60% más rápido que la competencia. Además, la nueva generación ofrecía más del doble de rendimiento que la generación anterior de servidores basados en el procesador Intel Xeon, que era capaz de ejecutar aplicaciones de 32 y 64 bits. Aparece Intel Core *2* Quad, una serie de procesadores con 4 núcleos, asegurando ser un 65% más rápidos que los Core 2Duo disponibles anteriormente. Para poder crear este procesador se tuvo que incluir 2 núcleos Core bajo un mismo empaque y comunicarlos mediante el bus del Sistema, para así totalizar 4 núcleos reales.

En lo relativo al mundo de los dispositivos móviles, Intel presenta en 2008 la familia de procesadores Intel Atom con su nueva tecnología de integración de 45 nm CMOS.

No es hasta el año 2010 que Intel comienza la fabricación masiva de sus procesadores Core. Intel Core i3, i5 e i7 de cuatro núcleos. Las continuas evoluciones de estos procesadores comprenden las micro arquitecturas Nehalem, Sandy Bridge, Ivy Bridge y Haswell. Poseen la tecnología Hyper Threading que permite que cada núcleo del procesador trabaje en dos tareas al mismo tiempo, ofreciendo el rendimiento que el usuario necesita para ejecutar varias tareas a la vez. En tanto,

Intel Core i5 e Intel Core i7 integran Turbo Boost, que incrementa de forma automática la velocidad de procesamiento de los núcleos por encima de la frecuencia operativa básica si no se han alcanzado los límites especificados de energía, corriente y temperatura.

2.2.2 CARACTERISTICAS ACTUALES DE LA ARQUITECTURA X86 Y X64 EN INTEL

La arquitectura tiene un juego de instrucciones de longitud variable, por tanto nos encontramos ante una arquitectura tipo CISC (*Complex Instruction Set Computer*). Dada la retro compatibilidad que ha caracterizado históricamente a los procesadores fabricados por Intel, el juego de instrucciones actual deriva de los antiguos procesadores 8008 y 8080. Incluye acceso a memoria no alineado y en los procesadores actuales se cuenta con soporte para números enteros pequeños.

Durante la ejecución los procesadores x86 emplean algunos pasos extra para separar las instrucciones en microinstrucciones. Estas operaciones son realizadas por la unidad de control que se encarga de planificar el orden de ejecución de estas micro-instrucciones. El motivo principal es tratar de maximizar el pipeline. También permite ejecución fuera de orden. (8) Actualmente Intel ha desarrollado una estructura híbrida entre CISC y RISC para tratar de aprovechar mejor las operaciones de pipeline que este último juego de instrucciones ofrece. Con este avance trata de equipararse a ARM.

MODOS DE CPU

Un procesador que implemente la arquitectura x86 puede ejecutar en los siguientes modos:

- Modo Protegido: Modo normal de operación.
- Modo real: Modo restrictivo con limitaciones de acceso a memoria.
- System Management Mode (SMM): en el que es suspendida toda la ejecución normal (incluyendo el sistema operativo), y es ejecutado un software especial separado en un modo de alto privilegio.

Un procesador que implemente la extensión x64 puede operar a su vez en dos sub-modos distintos:

- Modo de Compatibilidad: permite la ejecución como si se estuviera ejecutando una aplicación en modo 32 bits.
- Modo 64-bit: Con todas las capacidades de la arquitectura de 64 bits.

REGISTROS EN X86 Y X64

La arquitectura x86 cuenta en general con la los siguientes tipos de registro.

8 Registros de 16 bits de propósito general:

- Acumulador: Usado para operaciones aritméticas
- Base: Usado como puntero a los datos
- Datos: Para operaciones aritméticas y operaciones de entrada salida
- Contador: Para bucles y operaciones de desplazamiento
- Puntero de pila
- Puntero a la base de la pila
- Índice al origen: para apuntar al origen del flujo de operaciones.
- Índice al destino: para apuntar al final del flujo de operaciones.

Todos los registros pueden ser accedidos en modo 16 y 32 bits. Están diferenciados por la nomenclatura del registro. En los procesadores de 64 bits la nomenclatura cambia para denotar que se está accediendo a un registro de 64 bits.

También es posible direccionar los primeros cuatro registros en mitades de 8 bits, principalmente para facilitar la retro compatibilidad con programas desarrollado para versiones antiguas de los procesadores desarrollados por Intel.

6 registros para segmentación:

- A pila: apunta a la pila.
- Puntero al código del programa: apunta al principio del código del programa.
- Puntero al segmento de datos.
- Extra: Para datos extra.
- FS: Para más datos externos.
- G: Para incluso más datos extra.

Muchas aplicaciones de los sistemas operativos modernos usan un modelo de memoria que apunta a casi todos los segmentos de memoria y usan la paginación.

Registros EFLAGS

Registros de 32 bits que almacenan valores booleanos, resultados de operaciones y el estado del procesador.

JUEGO DE INSTRUCCIONES X86 Y X64

Dado que nos encontramos ante una arquitectura con juego de instrucciones CISC[1] el juego de instrucciones es amplísimo y ha evolucionado con cada nuevo desarrollo de los procesadores Intel, por tanto, si se desea conocer en profundidad el juego de instrucciones se detalla en la bibliografía el enlace a la documentación que Intel facilita gratuitamente (8) y que detalla una a una las operaciones que se permiten en la máquina. De modo representativo pueden ser agrupadas en 7 tipos diferentes de instrucciones:

- De transferencia de datos: Utilizadas para movimiento interno de datos, swap etc.
- De control de flujo: Saltos condicionales, incondicionales, por desbordamiento, en caso de ser igual a cero, instrucciones para bucles y en general soporte para estructuras de control del flujo del programa.
- Aritméticas: Toman dos operandos y aplican las operaciones sobre un registro o memoria. Se contemplan operaciones de suma, resta, multiplicación, división, operaciones con acarreo, incrementos y decrementos y aritmética de punteros.
- Lógicas: operaciones sobre bit a nivel lógico, operaciones como and, *or*, *xor*, *not*.
- Desplazamiento: Incluyen instrucciones de desplazamiento a nivel de registro y memoria.
- Interrupciones: Incluyen las instrucciones y rutinas especiales para el tratamiento de interrupciones y llamadas al sistema. Se utilizan para cambiar los estados del procesador en función de la instrucción solicitada. Gestiona tanto interrupciones software como interrupciones hardware y excepciones.
- Otras: como incrementos operaciones sobre la pila, alteración de los *flags* de un registro e instrucciones de entrada salida.

2.2.3 RELEVANCIA PARA EL PROYECTO DE LA ARQUITECTURA INTEL X86 X64

A efectos de este estudio, interesa conocer si una arquitectura basada en un juego de instrucciones CISC puede competir a nivel energético con una arquitectura más simple, enfocada al ahorro energético. Al reducir el número de instrucciones que son necesarias se espera obtener datos relevantes del uso de recursos hardware específico.

Intel ofrece el perfecto marco de comparación principalmente porque prácticamente el estándar de los ordenadores de sobremesa. Es el fabricante referente y el enemigo a batir por los principales fabricantes.

[1] Actualmente Intel utiliza un juego de instrucciones reducido para la gestión de microinstrucciones pero a efectos prácticos dado que el soporte no es completo, lo consideraremos CISC.

2.3 COMPARATIVA DE AMBAS ARQUITECTURAS

Pese a que tanto ARM como la arquitectura empleada por Intel son muy distintas entre sí, ambas han demostrado a lo largo de su historia ser muy buenas en sus respectivos campos. Como se puede apreciar a simple vista el uso de dos tipos diferentes de juegos de instrucciones es la principal diferencia en ambas arquitecturas.

ARM emplea RISC al tener una estructura e instrucciones más sencillas, poseen instrucciones de tamaño fijo con pocos formatos y sólo las instrucciones de carga y almacenamiento acceden a la memoria de datos. El objetivo al diseñar máquinas con esta arquitectura es facilitar el paralelismo en la ejecución de instrucciones y permitir realizar tareas menores con procesos más cortos lo que al final conlleva una disminución de la energía empleada. Además las instrucciones se pueden implantar por hardware directamente en la CPU, lo cual elimina el micro código y la necesidad de decodificar instrucciones complejas. Otro factor relevante es la reducción de accesos a memoria únicamente a las instrucciones load y store. Otro factor relevante es la alta capacidad para Pipeline que se consigue con un juego reducido de instrucciones al disminuir la etapa de decodificación al no haber instrucciones complejas. Como principal desventaja contamos con que la programación se complica dado que el nivel de abstracción es considerablemente menor que en las arquitecturas que emplean CISC.

Sin embargo, son los procesadores x86 los que destacan en rendimiento. Esa arquitectura más compleja permite que se apliquen más optimizaciones que se hacen mientras se ejecuta la aplicación, como por ejemplo el intercambio de orden de instrucciones para mejorar el tiempo de ejecución. Como desventaja podemos reseñar también de la necesidad de más espacio físico y mayor consumo de energía.

Estamos por tanto ante una competencia entre dos modelos del juego de instrucciones, RISC y CISC, pese a que Intel implemente actualmente un juego de instrucciones reducido para el manejo de las microinstrucciones (9). Este estudio pretende arrojar luz eliminando el factor tiempo y tratando de averiguar si la arquitectura ARM puede competir en consumo energético con un procesador x86 de Intel.

Ambos fabricantes tratan actualmente de acercarse el uno al otro, ARM trata de aumentar el rendimiento sin sacrificar consumo energético, mientras que Intel trata de acercarse al consumo de ARM para poder irrumpir en el mercado de los dispositivos móviles.

A nivel técnico tanto los registros como como el juego de instrucciones están optimizados para el propósito general de la arquitectura. Por tanto, a priori solamente podemos decir que se espera que Intel se postule favorita en cuanto a velocidad de ejecución, mientras que ARM despunte en cuanto a consumo energético.

2.3.1 MODELOS DE COMPUTACIÓN PARALELA

Paralelizar, a grandes rasgos es reducir un problema en subproblemas capaces de ser resueltos de forma concurrente. En el ámbito de la computación dado el cambio de paradigma hacia arquitecturas multinúcleo, este modelo supone una mejora de rendimiento, reduciendo el tiempo de ejecución al dividir las tareas entre los diferentes núcleos de procesamiento. (10)

Se han postulado diferentes técnicas para reducir los problemas en unidades procesables concurrentemente:

- A nivel de Instrucción: Se ejecutan las instrucciones paralelamente, sin cambiar el resultado del programa. El pipeline de instrucciones es el perfecto ejemplo.
- A nivel de Datos: Cuando los datos se distribuyen entre diferentes procesadores para ser procesados y unidos posteriormente.
- A nivel de Tareas: Cuando cálculos completamente diferentes son repartidos entre los distintos nodos de procesamiento en forma de tareas a realizar.

En líneas generales la memoria de un computador es compartida entre todos los elementos de procesamiento con su propio espacio de direcciones, distribuido lógicamente. Contamos con dos tipos de acceso a dicha memoria:

- UMA: si se equipara en latencia y ancho de banda a las diferentes unidades de procesamiento.
- NUMA: cuando el acceso a regiones de memoria de otra unidad de proceso se realiza a menor latencia que la asignada a la propia. Deriva en un uso más eficiente de las memorias cache de cada unidad.

Las estrategias de paralelización están fuertemente ligadas al número de procesadores disponibles en la máquina concreta, puesto que habitualmente se tiende a separar las tareas por hilos de ejecución asignados a un procesador concreto.

Otra técnica ampliamente utilizada en procesadores de nueva generación es la incorporación del *multithreading*. Esta técnica permite asignar a un mismo procesador varios hilos de ejecución de forma que se puedan implementar técnicas como el pipeline entre diferentes hilos de ejecución.

La aceleración producida por aplicar técnicas de paralelización se debe de conocer el tiempo que tarda la aplicación secuencial y la versión paralela:

$$Aceleración = \frac{Secuencial\ (s)}{Paralelo(s)}$$

Por contrapartida a mayor número de procesadores no implica necesariamente un aumento de la velocidad si la carga de trabajo no está lo suficientemente repartida o si el rendimiento de cada

procesador no es grande. Como se puede deducir de la conocida Ley de Amdahl Idealmente doblar el número de elementos de procesamiento debe de reducir a la mitad el tiempo de ejecución. Al doblarlo de nuevo se debe de poder reducir a la mitad. Esto no pasa prácticamente para ningún algoritmo. En breve resumen, el límite de elementos paralelizables lo establece la menor porción de código no paralelizable.

$$S = \frac{1}{\alpha} \qquad = \lim_{P \to \infty} \frac{1}{\frac{1-\alpha}{P} + \alpha}$$

Figura 7 Ley de Amdahl

Siendo alpha el porcentaje de tiempo que un programa emplea en operaciones secuenciales y P el porcentaje paralelo.

Si añadimos a la limitación producida por el código no paralelo, la computación paralela se enfrenta al problema de compartir información entre hilos. A menudo será necesario comunicar información y depender de la información de otros hilos. Este fenómeno nos hace acometer el problema de las condiciones de carrera, el bloqueo de variables (exclusión mutua) y por tanto el uso de secciones críticas que limitan aún más la capacidad de paralelización de los algoritmos.

Otro factor determinante a la hora de pensar en la paralelización es la granularidad del problema, teniendo en cuenta este factor, el paralelismo se puede caracterizar en:

- Grano fino: si las subtareas sincronizan habitualmente
- Grano grueso: si no se produce con tanta frecuencia.

A continuación pasaremos a describir las diferentes bibliotecas de paralelización que se emplean actualmente y que implementan estas técnicas.

- En primer lugar analizaremos el API de programación paralela que se ofrece en los sistemas operativos tipo *Posix*. *Pthreads*.
- Después analizaremos el API portable *OpenMP*, que ha sido adoptado como prácticamente un estándar para el desarrollo de aplicaciones con secciones de programación paralela.
- Por último realizaremos una breve descripción de la biblioteca de plantillas de programación paralela *Intel Threading Building Blocks* (*TBB*).

Se desarrollan estos tres APIs por su fuerte relación con la herramienta de benchmarking Parsec 2.1

PTHREADS

La implementación *POSIX Threads* (11) es el API estándar definido para crear y manipular hilos, define una serie de tipos, constantes y una amplia gama de funciones destinadas a implementar funciones paralelas. Los *threads* comparten totalmente la memoria entre ellos. A pesar de compartir memoria los hilos cuentan con información propia como el contador de programa, la pila y el valor de los registros de estado. Las funciones que facilita el API ofrecen mecanismos de *mutex y* sincronización mediante semáforos.

Dentro de los *pthreads* se definen dos tipos de hilos, a nivel usuario y a nivel de núcleo. En el primero la gestión es a nivel de aplicación, en otras palabras, gestionados por el programador. Esto presenta una serie de ventajas:

- No es necesaria la intervención del núcleo en modo privilegiado al estar alojados en el espacio de memoria de usuario.
- Se pueden planificar específicamente.

Así mismo tienen una serie de desventajas:

- Cuando un hilo realiza una llamada al sistema quedan todos bloqueados.
- No se puede asignar más de un hilo a un procesador puesto que el núcleo no interviene.

A nivel de núcleo el propio sistema operativo realiza la gestión de dichos hilos lo que aporta una serie de ventajas:

- El *kernel* puede planificar simultáneamente múltiples hilos del mismo proceso en múltiples procesadores.
- Si se bloquea un hilo, puede planificar otro del mismo proceso.
- Las propias funciones del *kernel* pueden ser multihilo.

Como principal desventaja el cambio de un hilo a otro requiere de cambio a modo privilegiado.

Ya que *pthreads* es una biblioteca *POSIX*, se podrán portar los programas hechos con ellos a cualquier sistema operativo *POSIX* que soporte *threads*.

OPEN MP

Figura 8 Logotipo de OpenMP

La interfaz de aplicaciones *OpenMP* (12) proporciona igualmente una biblioteca de paralelismo a nivel de tarea. Mediante directivas del compilador, rutinas y variables de entorno para mejorar el tiempo de ejecución.

Usa un modelo escalable y portable basado en la estrategia *fork-join* que se resume en el uso de un hilo maestro que al llegar a una directiva divide la carga de trabajo entre varios hilos. Al terminar la ejecución de la zona paralela es el propio hilo maestro el encargado de destruir el conjunto creado de hilos.

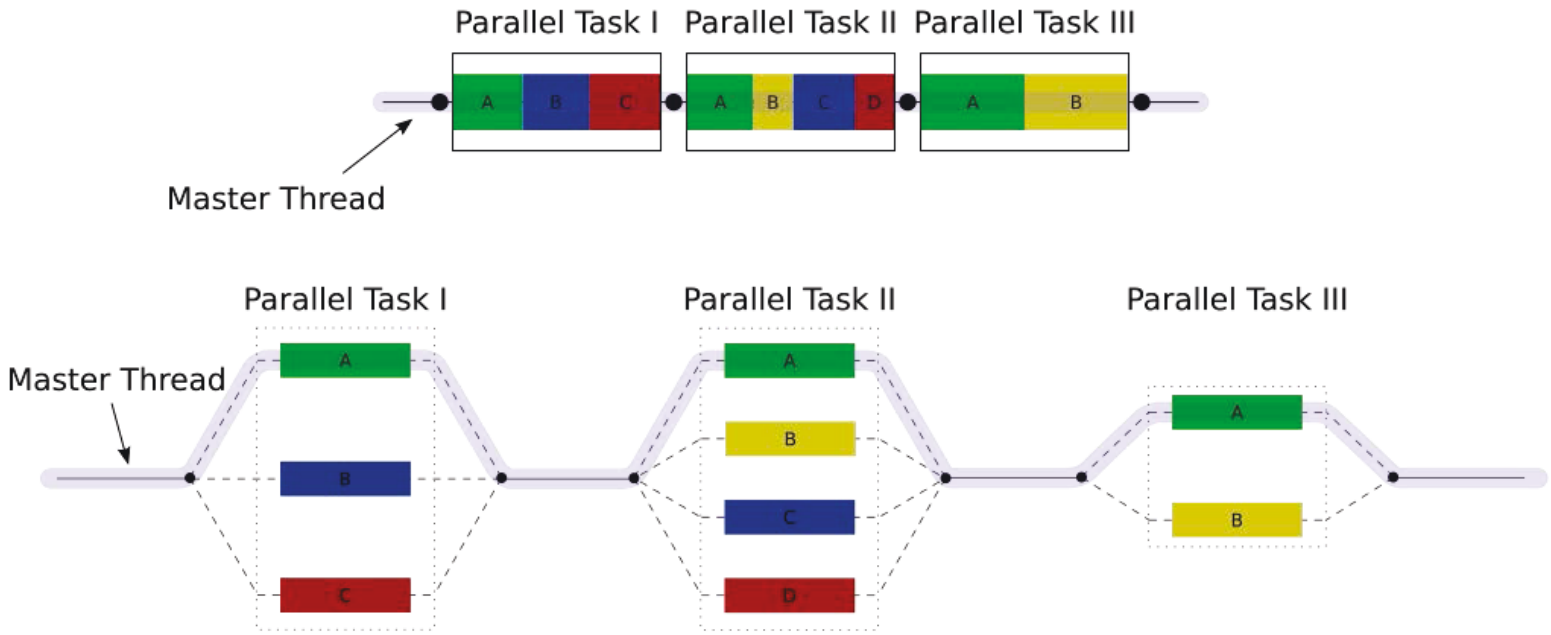

Figura 9 Proceso de creación y destrucción de hilos en OpenMP

Para realizar este proceso la zona paralela debe ser marcada apropiadamente empleando las funciones que la biblioteca ofrece. Los hilos ejecutan de forma independiente pudiendo indicarse si han de ser usados a nivel de tarea tanto a nivel de datos como por tareas. Las funciones más destacables que ofrece *OpenMP* están relacionadas con dos principales técnicas:

- Paralelizar bucles asignando un índice del bucle a cada hilo, obteniendo una granularidad fina para este tipo de secciones.
- Especificando regiones paralelas, distribuyendo la carga de forma explícita obteniendo paralelismo de granularidad gruesa.

Además permite modificar las variables de entorno del *API* para cambiar el comportamiento de las secciones paralelas. Especificar explícitamente que variables son compartidas, que variable se utiliza para operaciones de reducción sobre variables.

En lo referente a la gestión de gestiones críticas, condiciones de carrera y operaciones atómicas. *OpenMP* permite definir en él código que tipo de directivas de sincronización se debe aplicar en cada caso.

Para la gestión de los hilos permite especificar si se debe realizar la gestión de forma dinámica o definida por el usuario. Así mismo permite definir el uso de paralelismo anidado (*Paralelismo sobre regiones ya paralelizadas*).

La única desventaja que presenta es no poder definir la afinidad de los hilos a un procesador determinado. Este dato no facilita el acceso a memoria en arquitecturas con acceso a memoria no uniforme.

En sus últimas versiones ofrece soporte para tarjetas gráficas y soporte para procesadores vectoriales incluyendo soporte para instrucciones simples con múltiples datos.

INTEL TBB

Threading Building Blocks (13) es la biblioteca de plantillas de programación paralela a nivel de tareas desarrollado por Intel. Su principal premisa es el alto nivel de abstracción. Un programa *TBB* crea, sincroniza, asocia y destruye los hilos que asocia a una tarea determinada. La división del trabajo a nivel de tarea se realiza asignando pequeñas tareas a los hilos disponibles.

Implementa la técnica conocida como *robo de tareas* para balancear la carga de trabajo en los núcleos del procesador. De este modo, una vez repartidas las tareas entre los diversos procesadores una tarea terminada libera un hilo, al cual puede ser asignada parte de la carga de otro de los hilos en ejecución. Este robo de tareas permite mantener ocupado el máximo número de procesadores disponibles el máximo tiempo posible. A diferencia con otras bibliotecas de paralelismo, TBB crea todos los hilos disponibles. De este modo siempre están disponibles durante toda la ejecución del programa. Esto aporta la ventaja de la escalabilidad de las aplicaciones sin mucha atención por parte del programador, puesto que el código no se tiene que alterar para incluir más hilos.

Dado que enfatiza la paralelización a nivel de datos escala muy bien con conjuntos de datos muy grandes y muchos procesadores al realizar el mencionado balanceo de carga.

Su programación es bastante más sencilla que la mayoría de bibliotecas de paralelismo puesto que cuenta con plantillas. En caso de que el problema no se adapte a las tareas que define, permite generalas y gestionarlas.

Una de las características más curiosas de *TBB* es que es compatible con las bibliotecas de paralelismo existentes como *Pthreads* u *OpenMP*, por tanto es combinable con ellas.

2.4 HERRAMIENTAS DE BENCHMARKING

Un *benchmark* es una aplicación diseñada específicamente para probar las capacidades de un dispositivo empleando cargas de trabajo igual o similar a un proceso real con carga de trabajo real. El fin es medir de un modo controlado las verdaderas capacidades de él sistema. Su uso está realmente extendido en la actualidad dada la ferviente competencia entre los diferentes fabricantes por ofrecer productos más desarrollados, rápidos y eficientes que la competencia.

El benchmarking es una de las herramientas fundamentales para el estudio de una arquitectura completa, tradicionalmente el tiempo ha sido el instrumento más preciso para medir el rendimiento. La actual tendencia hacia las arquitecturas paralelas requiere de nuevos *benchmarks* que permitan cuantificar el rendimiento de las arquitecturas en situaciones en las que el paralelismo es el protagonista.

Actualmente son muchas las herramientas académicas y profesionales destinadas a medir el rendimiento de un sistema, pero normalmente suelen dedicarse únicamente a una pequeña porción de la arquitectura medida.

Lo completamente óptimo sería cubrir ampliamente las siguientes características deseables en una suite de *benchmarking*:

- Estar pensada para aplicaciones *multithread* en entornos de memoria compartida.
- Las cargas de trabajo deben estar adaptadas para sacar el rendimiento de los procesadores de nueva generación.
- Diversidad en las pruebas, compatibilidad con diversas plataformas y amplio abanico de programas que cubran todos los ámbitos de la computación, o su gran mayoría. Deben de cubrir las características más significativas del espacio de aplicaciones con las que se plantea medir.
- Uso de programas modernos, actuales y usados en situaciones reales.
- Mantenimiento e Investigación constante.

Además otras características deseables favorecen el desarrollo de este proyecto serían:

- Contar con un amplio conjunto de pruebas que permita medir el rendimiento de la mayoría de los aspectos relevantes de la arquitectura ARM y x86.
- Permitir ser lo suficientemente adaptable y libre para poder realizar modificaciones a la hora de adaptarlo a una arquitectura como ARM, que cuenta con pocos desarrollos y soporte en este campo.

En este sentido se han contemplado varias suites en busca de satisfacer estos requisitos:

- PARSEC 2.1: Destinada a *Multithreding* en entornos de memoria compartida.
- LINPACK: Centrada en solucionar ecuaciones y algebra linear. Las rutinas son en Fortran.
- SPLASH-2: Compuesta por aplicaciones *Multithreading* y memoria compartida.
- SPEC: En varias de sus variantes, referente en el mundo industrial, con una colección muy amplia de problemas.

Tras analizar las cargas de trabajo, los problemas que abarca y la caracterización de los mismos hemos descartado los siguientes benchmark por las siguientes razones:

- LINPACK: Muy específico, no aporta una visión general en entornos de trabajo reales y aplicaciones parecidas a las de alto rendimiento.
- SPLASH-2: No incluye modelos como Pipeline, las aplicaciones que cubre no han sido actualizadas con el tiempo y los algoritmos han quedado un poco obsoletos.
- SPEC: No incluye algunos de los modelos más comunes de problemas de programación paralela, como el de productor-consumidor. Es de licencia privada y poco adaptable dado su elevado número de pruebas.
- Otro conjunto de benchmarks han sido descartados únicamente por centrarse en un área concreta, no por ello dejan de ser herramientas útiles, simplemente por operatividad se prefiere una suite unificada y sólida en lugar de realizar nuestra propia suite compuesta por pruebas individuales.

Por tanto, se ha decidido que la suite que trataremos de adaptar sea PARSEC 2.1 de la universidad de Princeton puesto que la consideramos lo suficientemente completa, caracterizada, adaptada y paralela para estudiar en profundidad las capacidades de ARM.

2.4.1 PARSEC 2.1

Figura 10 Parsec Benchmark Suite

La suite de *benchmarking* PARSEC (*Princeton Application Repository for Shared-Memory Computers*) está desarrollada por la universidad de Princeton, en colaboración con empresas como Intel y diversas universidades a nivel mundial. Se ha postulado como un referente a nivel mundial entre las suites de benchmarking abiertas y accesibles a la comunidad de desarrolladores.

Se encentra por el momento en su tercera versión estable. Dada su naturaleza abierta, la suite tiene un amplio número de desarrolladores independientes que dan soporte mediante parches a las diversas versiones.

Parsec trata de aunar por un lado una amplia selección de programas con la intención de que sean lo suficientemente diversos para poder ser representativo en estudios académicos. Cuenta con 10 aplicaciones y 3 "*kernels*". Tratan de cubrir diferentes características de las cargas de trabajo y combinarlas para abarcar todas las principales técnicas de programación paralela. Por otro lado ofrece medidas de y rendimiento. (14)

Para cada programa, Parsec ofrece diferentes configuraciones:

	Paralelización		
Carga	**Pthreads**	**OpenMP**	**Intel TBB**
blackscholes	Si	Si	Si
bodytrack	Si	Si	Si
canneal	Si	No	No
dedup	Si	No	No
facesim	Si	No	No
ferret	Si	No	No
fluidanimate	No	No	Si
freqmine	Si	Si	No
raytrace	Si	No	No
streamcluster	Si	No	Si
swaptions	Si	No	Si
vips	Si	No	No
x264	Si	No	No

Tabla 1 Modelos de paralelización soportados por cada prueba

Todas las pruebas soportan las versiones del compilador *GCC* en sus versiones 4.2 y 4.3, así como Icc 10.1 de Intel. Se da soporte completo para Linux y Solaris y parcial para Windows. El soporte a las principales arquitecturas se da únicamente en arquitecturas x86, x86_64 y Sparc. Por tanto no está garantizada la portabilidad para arquitecturas como ARM.

La siguiente tabla muestra la caracterización de cada prueba, atendiendo al uso previsto de cada una. También se define el modelo de paralelismo modelos de programación paralela que se ha empleado. Se trata de abarcar la mayor casuística respecto a granularidad del problema. Así mismo se abarca la máxima casuística respecto a la comunicación y la compartición de datos.

		Paralelización			Uso de datos	
Programa	**Dominio de Aplicación**	**Modelo**	**Granularidad**	**Conjunto de Trabajo**	**Compartición**	**Intercambio**
blackscholes	Análisis Financiero	Datos paralelos	Gruesa	Pequeña	Baja	Baja
bodytrack	Visión por computador	Datos paralelos	Media	Media	Alta	Media
canneal	Ingeniería	No estructurado	Fino	Ilimitado	Alta	Alta
dedup	Almacenaje corporativo	Pipeline	Media	Ilimitado	Alta	Alta
facesim	Animación	Datos paralelos	Gruesa	Largo	Baja	Media
ferret	Búsqueda de similitudes	Pipeline	Media	Ilimitado	Alta	Alta
fluidanimate	Animación	Datos paralelos	Fina	Largo	Baja	Media
freqmine	Minería de datos	Datos paralelos	Media	Ilimitado	Alta	Media
raytrace	Visualización	Datos Paralelos	Media	Largo	Media	Media
streamcluster	Minería de datos	Datos paralelos	Media	Medio	Baja	Media
swaptions	Análisis financiero	Datos paralelos	Gruesa	Medio	Baja	Baja
vips	Procesamiento multimedia	Datos paralelos	Gruesa	Medio	Baja	Media
x264	Procesamiento multimedia	Pipeline	Gruesa	Medio	Alta	Alta

Tabla 2 Relación de pruebas de Parsec y caracterización de las cargas de trabajo.

Explicaremos el problema que plantean y resuelven en el apartado de pruebas.

TAMAÑO DE LAS CARGAS PARA CADA TEST

Parsec define 6 tipos de carga de trabajo para cada una de sus aplicaciones de Test:

- Test: Muy pequeña, para comprobar funcionalidad en casos básicos.
- *Simdev*: Carga muy pequeña y pensada para realizar pruebas de desarrollo.
- *Simsmall*, *simmedium* y *simlarge*: Cargas para simulación pequeña, mediana y grande respectivamente.
- *Native*: Carga de trabajo real. Destinada a probar intensamente todas y cada una de las estructuras de paralelismo.

2.4.2 PERF

Perf es el conjunto de herramientas de monitorización de recursos que ofrece el *kernel* de Linux. (15) Esta herramienta es capaz de monitorizar los recursos que se están empleando durante la ejecución de un programa. Es una herramienta muy potente a la hora de monitorizar el rendimiento que registra el hardware.

Algunas de las características más interesantes que ofrece Perf es lo ligero que es, no interfiriendo en el rendimiento o haciéndolo en una porción prácticamente despreciable de los resultados. Además estas medidas son contabilizadas en todas las CPUs por tanto se convierte en una herramienta capaz de evaluar el rendimiento general en programas que hagan uso de una arquitectura multinúcleo.

Dispone de diferentes opciones de monitorización:

- perf stat: obtiene monitorización de los eventos.
- perf record: almacena estadísticas durante un conjunto de pruebas empleando *stat*.
- perf report: separa la monitorización por evento, proceso y función.
- perf annotate: anota el ensamblado del código fuente y lo relaciona con los eventos monitorizados.
- perf top: Monitorización en tiempo real de los eventos y los contadores
- perf bench: Ejecuta *microbechmarks* sobre cada uno de los eventos.

Sin duda el apartado más útil es *stat* a nivel de monitorización de una aplicación. Stat permite monitorizar los siguientes eventos hardware:

- Eventos hardware: tales como ciclos, instrucciones, referencias a las caches, instrucciones de saltos y sus correspondientes fallos, ciclos en el bus de datos.
- Eventos hardware concretos de la caché: cargas, almacenamiento, captaciones (*fetch* y *prefetch*)
- Eventos software: como fallos de paginación, cambios de contexto, cambios de CPU.
- Eventos hardware del planificador.

3. EXPERIMENTACIÓN

En este apartado detallaremos el proceso seguido para abordar el problema, la tecnología que se ha decidido emplear, la implementación final de la solución, las pruebas realizadas y los resultados obtenidos.

3.1 PLANTEAMIENTO DEL PROBLEMA

Como se ha comentado brevemente en los objetivos generales de este proyecto se pretende comparar la arquitectura x86 y la arquitectura ARM a nivel de rendimiento y consumo energético.

Para ello, trataremos de portar a la arquitectura ARM un *benchmark* probado a fondo en la arquitectura Intel, que sea lo suficientemente completo, esté bien caracterizado y cubra suficientemente las diferentes API's de paralelismo que se han contemplado en el estado del arte.

Esto deriva en un estudio de las diferentes opciones a la hora de realizar la compilación en ARM, dadas las capacidades a priori más reducidas de esta arquitectura, existen varias posibilidades a la hora de compilar los ficheros fuentes del benchmark seleccionado. Se puede optar por realizar compilación cruzada o directa. Justificaremos en el siguiente apartado la decisión adoptada.

Una vez realizada la compilación y las pruebas de funcionalidad básicas pasaremos al apartado de pruebas. De este modo tendremos medidas fiables de las verdaderas capacidades que ambas arquitecturas enfrentándose al mismo problema. Con estos resultados, calcularemos el coste energético derivado del tiempo de uso de ambas arquitecturas.

El proyecto pretende hacer especial énfasis en el paralelismo calculando el speed-up que se produce variando el API empleado para ejecutar las pruebas.

3.2 HARDWARE Y SOFTWARE EMPLEADO

Este apartado describe toda la tecnología empleada, justificando su elección y evaluando los problemas derivados de su uso. En resumen se ha empleado:

- Como representante de la arquitectura x86 emplearemos un procesador Intel Core i5-2500 CPU 3.30GHz con 4 núcleos de procesamiento, bajo un sistema operativo Linux Ubuntu 13.
- Para la arquitectura ARM disponemos de un procesador Samsung Exynos4412 Cortex-A9 Quad Core 1.4Ghz con 4 núcleos de procesamiento, bajo un sistema operativo Linux Linaro.
- Para realizar las pruebas las aplicaciones contenidas en Parsec versión 2.1.
- Para la compilación de las pruebas emplearemos el compilador GCC en su versión 4.6.
- Para la paralelización emplearemos las siguientes tecnologías:
  - *Posix Threads*
  - *OpenMP*
  - *Intel TBB*
- Para la monitorización de los recursos en ambos procesadores emplearemos la función ofrecida en el *kernel* de Linux, Perf.
- Para la medición del consumo energético recurriremos a las especificaciones técnicas de los fabricantes.
- Para la captura de los datos obtenidos una aplicación Java diseñada exprofeso para extraer los datos que genera Perf.

3.2.1 INTEL CORE I5-2500

Se ha decidido emplear como representante de la arquitectura x86 x86-64 un procesador Intel Core i5-2500 que cuenta con las siguientes características (16):

- Pertenece a la segunda generación de procesadores Intel Core de 32 nm, contando con la tecnología Sandy Bridge.
- Tiene una velocidad de reloj de 3,3 Ghz ampliable a 3,7 Ghz con la tecnología Turbo Boost.
- Cuenta con 4 núcleos y cuatro hilos de ejecución, no cuenta con tecnología de *Hyper-Threading*.
- Cuenta con 4Gb de memoria RAM.
- Cuenta con una caché de 6Mb en dos niveles.
- Un juego de instrucciones de 64 bits.
- Un sistema operativo Ubuntu versión 13
- Un consumo de energía especificado máximo de 95 vatios.

La decisión de emplear este procesador se basa en primer lugar a la disponibilidad necesaria a la hora de ejecutar grandes baterías de pruebas. Dado que pertenece a la universidad está disponible

24 horas.

Representa fielmente la arquitectura x86 y está ampliado para dar soporte a la arquitectura de 64bits. El juego de instrucciones a su vez da soporte a ambas arquitecturas, teniendo un juego de instrucciones CISC y RISC para el procesamiento de microinstrucciones, esto nos da una aproximación más clara a la arquitectura ARM dado el uso de un juego de instrucciones parecido.

3.2.2 SAMSUNG EXYNOS4412 CORTEX-A9

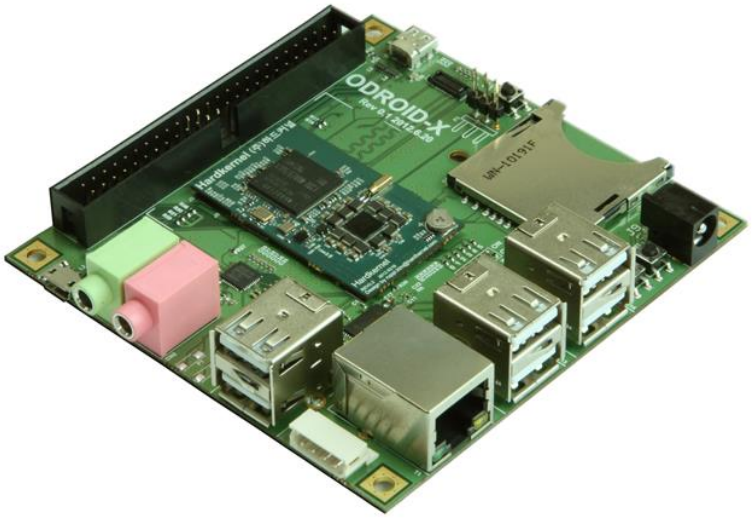

Figura 11 Equipo ARM usado para el proyecto.

El modelo empleado para realizar las pruebas en la arquitectura ARM es un procesador Samsung Exynos4412 Cortex-A9 Quad Core 1.4Ghz ensamblado por la empresa *ODROID*, que distribuye el procesador junto con los conectores necesarios para agregar periféricos.

Cuenta con las siguientes características:

- 4 procesadores Cortex-A9 de 1.4Ghz.
- 1 GB de memoria RAM.
- 1 MB de caché de nivel 2.
- Un juego de instrucciones de 32-bits.
- Un sistema operativo Linux Linaro.
- Un consumo de energía especificado máximo de 10 vatios.

La justificación de elegir esta máquina aparte de la disponibilidad al formar parte del hardware de la universidad es el juego de instrucciones *RISC*. El bajo consumo ha influido mucho en la elección. El uso de este modelo de procesador en toda la gama de móviles Samsung Galaxy SII, un modelo vendido ampliamente en todo el mundo y nos ayuda a escoger una máquina con bajo consumo que represente a los procesadores móviles.

3.2.3 PARSEC 2.1

Se ha escogido la versión 2.1 del paquete Parsec de *benchmarking*. Ha sido detallado en el estado del arte, por tanto solo justificaremos el porqué de este *benchmark*.

Cuenta con un amplio soporte por parte de la comunidad mediante parches y solución de problemas. Ha sido empleado ampliamente por la comunidad científica y en entornos académicos.

Está muy bien caracterizado y cubre una amplia gama de problemas reales. Por tanto es idóneo para medir equipos de propósito general. No cuenta con soporte para la arquitectura ARM, por tanto supone un reto a nivel técnico.

3.2.4 GCC 4.6

La elección de la versión 4.6 del compilador GCC se basa únicamente en el soporte común que tiene en ambas arquitecturas, pese a que Parsec no está garantizado para funcionar con la versión 4.6, se ha decidido intentar realizar la compilación con esta versión moderna del compilador.

En un primer momento se pensó que dadas las capacidades más reducidas de ARM la compilación se realizaría empleando compilación cruzada. Dado que se pretendía comprobar la verdadera capacidad de ARM en comparación con Intel, se pensó que sería interesante compilar nativamente las aplicaciones.

La compilación se ha realizado configurando el grado 3 de optimizaciones que ofrece GCC para todas las aplicaciones.

3.2.5 PROGRAMACIÓN PARALELA

Se ha decidido emplear las tecnologías de paralelización *PThreads*, *OpenMP* e Intel *TBB* dado que Parsec 2.1 ofrece versiones de sus programas usando dichas tecnologías (Véase la Tabla 1). Pese a que las dos primeras son soportadas por la arquitectura ARM, Intel TBB supone un reto, puesto que no se garantiza que funcione en la arquitectura.

Interesa conocer cómo se comportan las dos arquitecturas con las técnicas de paralelización propuestas. Especialmente *OpenMp* y *TBB* dado que interesa comparar ambos modelos de programación paralela a efectos de analizar, sobre una arquitectura móvil, que técnica funciona mejor.

3.2.6 PERF

La justificación del uso de Perf para analizar los componentes de ambos procesadores a parte de la información que aporta Parsec es la ligereza que tiene. No altera prácticamente la carga de trabajo y aporta información muy relevante respecto al número de instrucciones ejecutadas. Perf nos aporta indicadores concretos de los eventos hardware del procesador. Se pretende comprobar si los datos obtenidos dan valor añadido en el análisis de las diferentes pruebas sobre ARM e Intel. El análisis de los datos obtenidos de por ejemplo la memoria caché pueden servir para detectar errores o comportamientos anómalos en el entorno de pruebas. Con su uso se pretende afianzar las conclusiones respecto a que arquitectura realiza una gestión mejor de los recursos hardware.

3.2.7 MEDICIÓN DE ENERGÍA

Inicialmente se pretendía utilizar Hardware externo para realizar las medidas de energía en tiempo de ejecución, pero finalmente no fue posible dada la imposibilidad de adquirir el equipamiento necesario. Después se trató de obtener software de medición de consumo como *Likwid*, pero no se consiguió portar la herramienta a ARM dado que está específicamente diseñada para mediciones en la arquitectura x86. Por tanto se ha consultado la documentación técnica de los fabricantes para obtener una estimación de la potencia consumida.

3.2.8 HERRAMIENTA JAVA PARA EXTRACCIÓN DE DATOS

Dada la cantidad de pruebas que se pretende realizar, es necesario poder gestionar y agrupar dicha información para extraer conclusiones fehacientes. Se ha optado por programar una pequeña herramienta que extraiga de los ficheros generados la información y realice las operaciones necesarias para presentarlos correctamente (medias, tabulación, organización).

3.3 IMPLEMENTACIÓN DE LA SOLUCIÓN

Durante este capítulo se describirán los pasos concretos que se han realizado para establecer el entorno de pruebas, los problemas encontrados y la solución adoptada para solventarlos.

3.3.1 INSTALACIÓN EN INTEL

Presentamos primero la instalación en Intel dado que no supone ningún problema y únicamente se realiza siguiendo el manual que la documentación de Parsec aporta. Únicamente cabe reseñar que es conveniente aplicar los parches que la comunidad ha ido recopilando a lo largo de la vida de la versión 2.1 para conseguir subsanar pequeños fallos de funcionalidad. Después únicamente queda compilar todos los paquetes empleando la herramienta que acompaña a la distribución (*parsecmgmt*) especificando que se desea compilar todos los paquetes de aplicaciones y *kernels*.

Este proceso se debe realizar para cada técnica de paralelización. Obteniendo por tanto 4 versiones de cada programa y *kernel*. Una distribución secuencial (*gcc*), una versión paralelizada con Hilos (*phtreads*), otra con el API *OpenMp* (*openmp*) y por último una con *IntelTBB* (*tbb*).

La herramienta de gestión de la suite compila todo lo necesario para el modelo de paralelización deseado siempre que este esté disponible en la distribución. Además todas las bibliotecas necesarias para el funcionamiento de los *benchmarks* son compiladas siempre.

3.3.2 INSTALACIÓN DE PARSEC 2.1 EN ARM

Se ha realizado una copia de la distribución Parsec 2.1 tal y como se describe en el apartado anterior teniendo en cuenta las siguientes modificaciones:

- Ser compilan primeramente las herramientas “parsecmgmt -a build -p tools”
- Se incluyen, si no se tenían en la distribución Linaro los siguientes paquetes:
    - libX11
    - libXmu
    - libXext
    - libxcb
    - xproto
    - xextproto
    - xtrans
    - libpthread_stubs
    - libXau
    - kbproto
    - inputproto
    - jpeg

- La biblioteca TBB incluida en el propio benchmark se ha eliminado, siendo sustituida por una adaptada para ARM. En un primer momento se trató de usar la biblioteca que incluía la distribución. Tras no conseguir que tras compilarla funcionara, se optó por compilarla manualmente y agregarla a la biblioteca de funciones de Linux. De este modo al no encontrarse en el directorio Parsec, la herramienta trata de buscar los ficheros fuente en la carpeta *include* del *kernel*. La distribución TBB usada se basa en un proyecto localizado en el repositorio *GitHub*. Esta desarrollada por Simon Lynen en la siguiente dirección:
  - https://github.com/simonlynen/tbb-arm/
  - Este paso implica que las aplicaciones que incluyen TBB deben ser compiladas usando *parsecmgmt* de modo individual.
  - Los ficheros compilados deben ubicarse en la carpeta *include*.
- En el *kernel* canneal se necesita modificar dentro de los ficheros fuentes la biblioteca atomic.h agregando soporte para la arquitectura. Se puede obtener una versión adaptada aquí:
  - ftp://ftp.tw.freebsd.org/pub/FreeBSD-current/src/sys/arm/include/atomic.h
  - Una vez obtenido el fichero compilado se recomienda añadirlo a la carpeta *include.*
- Se recomienda tener actualizada la distribución Linaro para que cuente con las herramientas necesarias y actualizadas, en el desarrollo de este proyecto no hizo falta nada más para conseguir la compilación de los *benchmarks*. Es posible que al tratar de replicar este proceso se requieran más bibliotecas de funciones dependiendo del software instalado.

3.3.3 INCLUSIÓN DE PERF

Para poder añadir soporte para la monitorización de recursos, en primer lugar se debe de conseguir el paquete. En Intel se puede conseguir descargando el paquete de utilidades básico *linux-tools.* En ARM se puede instalar siguiendo los pasos que se describen en:

https://wiki.linaro.org/Platform/DevPlatform/Tools/Perf

Únicamente queda incluir perf dentro de la herramienta *parsecmgmt* añadiendo el comando apropiado como valor de la variable *default_submit* esto producirá que la salida de perf sea por la consola de comandos, no pudiendo ser derivada a un fichero, en nuestro caso interesaba desviar la salida a un fichero único. Para realizar dicha redirección y crear un fichero de salida se debe derivar en primer lugar la salida a */dev/null*. De esta forma la llamada a perf quedaría de la siguiente manera:

```
default_submit="3>nombre_del_fichero_salida  ruta_perf /perf stat -e cycles,instructions,cache-references,cache-
misses,branches,branch-misses,cpu-clock,task-clock,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-
store-misses,L1-icache-load-misses,branch-loads,branch-load-misses --log-fd 3 "

 submit=${default_submit}

 local bmexec_args=${run_args}

 eval cmd="\"${submit} ${bmexec} ${bmexec_args}\" > /dev/null;"
```

De este modo se consigue que *parsecmgmt* cada vez que se ejecute llame a perf para que comience a medir. Además, evita que se tomen medidas durante la carga y descompresión de los datos de prueba, solamente limitando la medida a la ejecución.

3.3.4 CÁLCULO DE LA ENERGÍA CONSUMIDA

Dado que únicamente se dispone de las medidas aportadas por los fabricantes de ambas máquinas expresadas en vatios por hora, una vez obtenidos los tiempos calcularemos el consumo energético con la siguiente ecuación:

$$\frac{Tiempo\ en\ segundos\ *\ Vatios/hora}{3600\ segundos\ en\ una\ hora}$$

Tomando como referencia para Intel 95 vatios (16) y 10 vatios para ARM[2] siendo para ambos casos el consumo máximo.

[2] Actualmente la página del fabricante ya no existe, se anexa una copia obtenida al comienzo de este proyecto de las medidas aportadas del fabricante.

3.4 PRUEBAS

En esta sección mostraremos los resultados obtenidos por los *benchmarks* en ambas arquitecturas acompañaremos los resultados de un breve análisis y comentario de los aspectos más relevantes observados para cada test. Tras los test realizaremos un análisis general de ambas arquitecturas.

Para cada uno de las aplicaciones disponibles en Parsec 2.1 se han realizado:

- 30 pruebas por cada modelo de programación, número de hilos (1, 2, 4, 8) y tamaño de la muestra (pequeño, mediano, grande, nativo).
- Se ha tenido en cuenta que no todas las pruebas están implementadas para todos los modelos de programación paralela.
- Se ha tenido en cuenta que ambas máquinas solo tienen 4 hilos de ejecución (no hay disponible tecnología como *Hyper-Threading* en Intel). Pese a ello se desea comprobar que ocurre al indicar que se gestionen las aplicaciones con 8 hilos.
- Se mostrarán los resultados en términos de tiempo (en segundos) y consumo energético (en vatios) tomando como referencia los valores de consumo máximo (95 vatios hora para Intel y 10 vatios hora ARM). También se facilitarán datos relativos al speed-up de las diferentes bibliotecas de paralelización que ofrece la suite Parsec.
- Si cabe reseñar algún aspecto relevante de algún evento adicional se comentará y se tratará de analizar.

Se ha obtenido la media de todos los resultados para los siguientes eventos del procesador:

- Ciclos de CPU: Interesa conocer si el número de ciclos es menor en ARM o Intel, de este modo podremos conocer que arquitectura emplea menos ciclos para ejecutar un programa.
- Instrucciones: El planteamiento es similar a los ciclos. De este modo podremos conocer que arquitectura emplea menos instrucciones, a priori consideramos que Intel debería de tener menor número de instrucciones, puesto que el juego de instrucciones que maneja es más amplio. Por tanto debería de ejecutar un menor número para realizar las mismas acciones que ARM cuyas instrucciones son más pequeñas, y por tanto requiere realizar más pasos para una misma acción.
- Referencias a cache y fallos: Interesa conocer qué arquitectura hace un uso mejor de las caches. Además gracias a Perf podremos conocer qué arquitectura falla menos.
- Saltos (*branches*) y fallos de salto: mismo motivo.
- Tiempo: Permitirá conocer que arquitectura tarda menos en ejecutar la prueba. A priori se piensa que Intel obtendrá mejores resultados dada la frecuencia de reloj, casi el doble que ARM y la cantidad de memoria RAM, cuatro veces mayor.

Pasamos a continuación a presentar los resultados obtenidos por las diferentes pruebas así como el consumo energético asociado a cada ejecución.

3.4.1 TEST BLACKSCHOLES

Aplicación de análisis financiero que calcula el portfolio de opciones Europeas sin riesgo. Realiza cálculos analíticos empleando el modelo de Black y Scholes. Para ello realiza una estimación mediante una ecuación de derivadas parciales que han de ser calculadas numéricamente. La carga de trabajo se distribuye con grano grueso y se realiza poca comunicación e intercambio en las versiones paralelas.

Se muestran a continuación las medidas del tiempo para el test en segundos:

Tamaño	Hilos	Secuencial		PThreads		OpenMP		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	0,5131	0,0716	0,5133	0,0740	0,5115	0,0652	0,8756	0,0689
	2			0,3690	0,0495	0,3825	0,0427	0,6144	0,0420
	4			0,3097	0,0309	0,3079	0,0316	0,4806	0,0320
	8			0,3129	0,0344	0,3494	0,0507	0,5314	0,0355
Simmedium	1	1,4321	0,2439	1,4495	0,2437	1,4349	0,2331	2,9223	0,2338
	2			0,9275	0,1498	0,9312	0,1385	1,8752	0,1380
	4			0,6895	0,0990	0,6704	0,0968	2,2787	0,0953
	8			0,6767	0,1003	0,7085	0,1465	2,0919	0,1040
Simlarge	1	5,1441	0,9096	5,1188	0,9074	5,1121	0,9004	11,8470	0,9027
	2			3,1010	0,5284	3,1145	0,5242	8,8630	0,5192
	4			2,1152	0,3332	2,1082	0,3399	9,5470	0,3370
	8			2,1179	0,3379	2,1879	0,4053	9,3480	0,3773
Native	1	958,3442	138,9778	961,1704	138,6880	964,8147	144,5710	900,6458	73,1158
	2			858,3529	79,7934	868,7843	82,2595	558,9629	40,3489
	4			651,2259	50,9929	646,3383	51,1009	399,9056	25,0604
	8			652,5563	49,7581	652,3902	51,1458	425,5314	25,9031

Tabla 1 Tiempos Test Blackscholes

Se puede apreciar que en todas las pruebas Intel ha obtenido unos resultados muy superiores en cuanto al rendimiento. Aproximadamente Intel tarda unas 7 veces menos que ARM. Pese a los tiempos se observan resultados curiosos si observamos las pruebas empleando las diferentes técnicas de paralelización. Para las pruebas secuenciales, con Pthreads y OpenMP para un solo hilo, los resultados son parecidos, mientras que con Intel TBB para cargas pequeñas el tiempo empeora. Por el contrario en cargas grandes el código tarda menos en ejecutarse. En cuanto a los ciclos, ARM tarda un orden de magnitud más (10^{12} frente a 10^{11} ciclos), respecto al número de instrucciones se observa que Intel también ejecuta menos, como se esperaba.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		PThreads		OpenMP		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1	1	1	1	1
	2			1,39	1,49	1,34	1,53	1,43	1,64
	4			1,66	2,39	1,66	2,06	1,82	2,15
	8			1,64	2,15	1,46	1,29	1,65	1,94
Simmedium	1	1	1	1	1	1	1	1	1
	2			1,56	1,63	1,54	1,68	1,56	1,69
	4			2,10	2,46	2,14	2,41	1,28	2,45
	8			2,14	2,43	2,03	1,59	1,40	2,25
Simlarge	1	1	1	1	1	1	1	1	1
	2			1,65	1,72	1,64	1,72	1,34	1,74
	4			2,42	2,72	2,42	2,65	1,24	2,68
	8			2,42	2,69	2,34	2,22	1,27	2,39
Native	1	1	1	1	1	1	1	1	1
	2			1,12	1,74	1,11	1,76	1,61	1,81
	4			1,48	2,72	1,49	2,83	2,25	2,92
	8			1,47	2,79	1,48	2,83	2,12	2,82

Tabla 2 Speed-Up Blackscholes

Se observa que TBB obtiene unos resultados muy buenos al aumentar el número de hilos en comparación a otras tecnologías. En lo referente al speed-up conseguido por con las diferentes APIs de paralelismo, observamos que TBB obtiene los mejores resultados en ambas máquinas, siendo mayor en Intel.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		PThreads		OpenMP		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	0,0014	0,0019	0,0014	0,0020	0,0014	0,0017	0,0024	0,0018
	2			0,0010	0,0013	0,0011	0,0011	0,0017	0,0011
	4			0,0009	0,0008	0,0009	0,0008	0,0013	0,0008
	8			0,0009	0,0009	0,0010	0,0013	0,0015	0,0009
Simmedium	1	0,0040	0,0064	0,0040	0,0064	0,0040	0,0062	0,0081	0,0062
	2			0,0026	0,0040	0,0026	0,0037	0,0052	0,0036
	4			0,0019	0,0026	0,0019	0,0026	0,0063	0,0025
	8			0,0019	0,0026	0,0020	0,0039	0,0058	0,0027
Simlarge	1	0,0143	0,0240	0,0142	0,0239	0,0142	0,0238	0,0329	0,0238
	2			0,0086	0,0139	0,0087	0,0138	0,0246	0,0137
	4			0,0059	0,0088	0,0059	0,0090	0,0265	0,0089
	8			0,0059	0,0089	0,0061	0,0107	0,0260	0,0100
Native	1	2,6621	3,6675	2,6699	3,6598	2,6800	3,8151	2,5018	1,9294
	2			2,3843	2,1057	2,4133	2,1707	1,5527	1,0648
	4			1,8090	1,3456	1,7954	1,3485	1,1108	0,6613
	8			1,8127	1,3131	1,8122	1,3497	1,1820	0,6836

Tabla 3 Energía consumida Blackscholes

Respecto al consumo energético podemos observar que ARM consume en general mucho menos que Intel, a excepción del modelo de paralelización Intel TBB. Este dato nos resulta cuando menos curioso. Por tanto, para este test, se considera que Intel es mejor que ARM en este tipo de aplicaciones siempre y cuando esté disponible una versión TBB del programa.

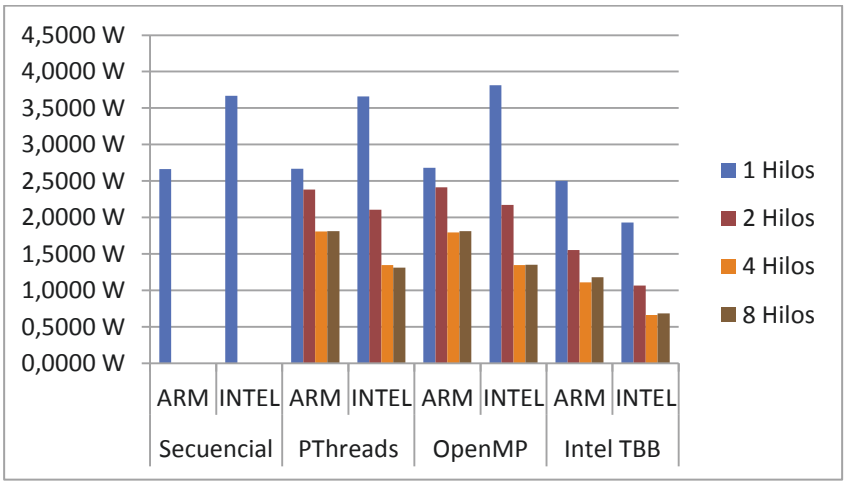

Figura 12 Consumo energético test Blackscholes con carga nativa

3.4.2 TEST BODYTRACK

Aplicación de visión por ordenador de Intel que trata de trazar el cuerpo humano a través de múltiples cámaras que toman secuencias de imagines. Especialmente destinado para probar las capacidades de los procesadores en aplicaciones de video vigilancia, animación e interfaces gráficas. La carga de trabajo se distribuye uniformemente entre los hilos, con alta comunicación entre los hilos en las versiones paralelas.

Se presentan a continuación los tiempos en segundos:

Tamaño	Hilos	Secuencial		PThreads		OpenMP		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	2,0503	0,2225	2,0166	0,2261	2,1889	0,2394	2,2429	0,2465
	2			1,4791	0,1364	1,5453	0,1390	1,5833	0,1443
	4			1,1662	0,0968	1,2395	0,0996	1,3090	0,0969
	8			1,1825	0,0988	1,2856	0,1305	1,3008	0,1031
Simmedium	1	5,1880	0,6658	4,9803	0,6678	5,5272	0,6535	5,6470	0,6653
	2			3,1492	0,3882	3,6462	0,3650	3,6149	0,3717
	4			2,3094	0,2305	2,6438	0,2419	2,5747	0,2334
	8			2,2226	0,2303	2,7582	0,2762	2,7002	0,2448
Simlarge	1	15,7742	2,2453	15,5253	2,2570	16,6930	2,1584	17,0655	2,1858
	2			8,9384	1,2586	10,0098	1,1699	9,9877	1,1881
	4			5,5565	0,7030	6,7379	0,7196	6,4945	0,6994
	8			5,5657	0,6935	6,8688	0,7856	6,5806	0,7533
Native	1	936,9308	135,0176	1019,1795	134,7790	1000,8684	128,2122	1009,6993	130,3305
	2			514,1928	75,0826	600,9961	71,1510	586,8773	71,1087
	4			315,9978	43,7316	408,9385	42,5419	383,7077	41,4680
	8			341,1109	42,6398	412,7646	47,4354	385,9414	43,6279

Tabla 4 Tiempos Test Bodytrack

A la luz de los datos podemos comprobar que Intel obtiene mejores resultados que ARM como se esperaba. En esta prueba no obtenemos resultados atípicos para las cargas de trabajo grandes. Podemos observar que para un hilo la versión secuencial se comporta mejor que el resto, como también cabía esperar.

Respecto a las herramientas de paralelización en este caso los mejores resultados los ha obtenido la versión Pthreads en ARM. Para Intel TBB ha funcionado mejor, obteniendo tiempos inferiores.

Respecto al resto de eventos del procesador, no se ha observado ningún dato relevante que merezca mención. Se repite el mismo patrón respecto al número de ciclos y las instrucciones que ambas arquitecturas llevan a cabo. ARM emplea más instrucciones y más ciclos que Intel.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		PThreads		OpenMP		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1	1	1	1	1
	2			1,36	1,66	1,42	1,72	1,42	1,71
	4			1,73	2,34	1,77	2,40	1,71	2,54
	8			1,71	2,29	1,70	1,83	1,72	2,39
Simmedium	1	1	1	1	1	1	1	1	1
	2			1,58	1,72	1,52	1,79	1,56	1,79
	4			2,16	2,90	2,09	2,70	2,19	2,85
	8			2,24	2,90	2,00	2,37	2,09	2,72
Simlarge	1	1	1	1	1	1	1	1	1
	2			1,74	1,79	1,67	1,84	1,71	1,84
	4			2,79	3,21	2,48	3,00	2,63	3,13
	8			2,79	3,25	2,43	2,75	2,59	2,90
Native	1	1	1	1	1	1	1	1	1
	2			1,98	1,80	1,67	1,80	1,72	1,83
	4			3,23	3,08	2,45	3,01	2,63	3,14
	8			2,99	3,16	2,42	2,70	2,62	2,99

Tabla 5 Speed-Up Bodytrack

A la vista de los datos relativos al speed-up conseguido, el mejor resultado lo obtiene ARM empleando la biblioteca Posix Threads para cuatro hilos de ejecución. TBB con Intel consigue un mejor rendimiento tanto con cargas pequeñas como con grandes. Estos datos nos indican que la gestión dada la alta comunicación, en ARM no se aprovecha correctamente el robo de tareas que ofrece TBB. Las esperas que se generan debido al rendimiento de los procesadores frenan la ejecución del programa produciendo más tiempo de espera que en Intel, donde se aprovecha mucho mejor la técnica del robo de tareas.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		PThreads		OpenMP		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	0,0057	0,0059	0,0056	0,0060	0,0061	0,0063	0,0062	0,0065
	2			0,0041	0,0036	0,0043	0,0037	0,0044	0,0038
	4			0,0032	0,0026	0,0034	0,0026	0,0036	0,0026
	8			0,0033	0,0026	0,0036	0,0034	0,0036	0,0027
Simmedium	1	0,0144	0,0176	0,0138	0,0176	0,0154	0,0172	0,0157	0,0176
	2			0,0087	0,0102	0,0101	0,0096	0,0100	0,0098
	4			0,0064	0,0061	0,0073	0,0064	0,0072	0,0062
	8			0,0062	0,0061	0,0077	0,0073	0,0075	0,0065
Simlarge	1	0,0438	0,0593	0,0431	0,0596	0,0464	0,0570	0,0474	0,0577
	2			0,0248	0,0332	0,0278	0,0309	0,0277	0,0314
	4			0,0154	0,0186	0,0187	0,0190	0,0180	0,0185
	8			0,0155	0,0183	0,0191	0,0207	0,0183	0,0199
Native	1	2,6026	3,5630	2,8311	3,5567	2,7802	3,3834	2,8047	3,4393
	2			1,4283	1,9813	1,6694	1,8776	1,6302	1,8765
	4			0,8778	1,1540	1,1359	1,1226	1,0659	1,0943
	8			0,9475	1,1252	1,1466	1,2518	1,0721	1,1513

Tabla 6 Energía consumida Bodytrack

En el apartado energético, respecto a la misma carga de trabajo ARM ha supuesto un ahorro energético en todas las pruebas. El mayor ahorro se ha obtenido empleando Pthreads con la configuración de 4 hilos. A la luz de los datos obtenidos se recomienda el uso de ARM en entornos de aplicación similares a Bodytrack siempre que el tiempo no sea un factor a tener en cuenta. Dado que bodytrack se usa para trazar el cuerpo humano a través de una secuencia de imágenes, ARM puede ser empleado para el análisis en diferido de dichas imágenes. Se recomienda el uso de una máquina más potente en caso de necesitar un aumento de la velocidad de cómputo.

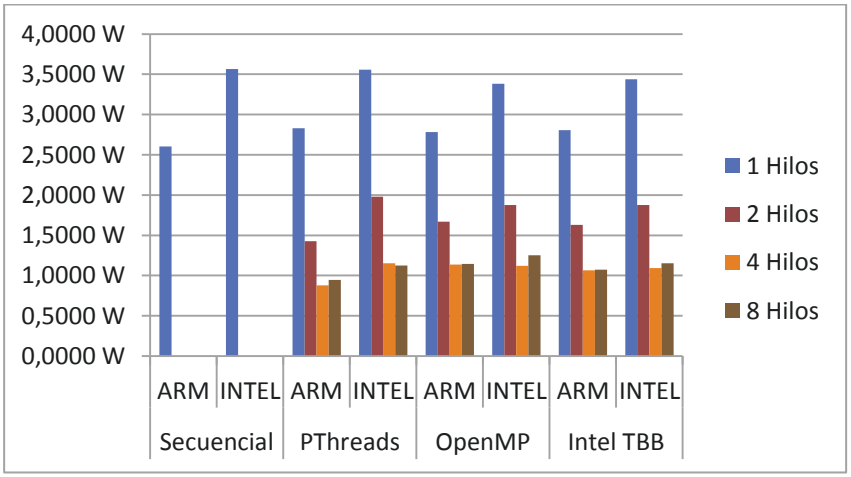

Figura 13 Consumo energético Test Bodytrack con carga nativa

3.4.3 TEST CANNEAL

Este *kernel* fue desarrollado por la Universidad de Princeton emplea un modelo de cache específica para minimizar el coste de diseño de microchips empleando el menor espacio disponible. La granularidad es realmente fina, puesto que el algoritmo no tiene prácticamente estructuras del tipo "*lock*" (cerrojos). La carga de trabajo por hilo es pequeña, pero por contrapartida la comunicación y el intercambio de información entre los hilos es alto.

Se muestran a continuación las medidas del tiempo para el test en segundos:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1		1,0261		1,0171
	2				0,9353
	4				0,9022
	8				0,9127
Simmedium	1		2,5944		2,5749
	2				2,2763
	4				2,1321
	8				2,1558
Simlarge	1		5,9961		6,0116
	2				5,2451
	4				4,8702
	8				4,9135
Native	1		157,3370		156,7254
	2				100,9176
	4				73,3321
	8				74,7086

Tabla 7 Tiempos Test Canneal

Dados los requisitos hardware del programa, los test en ARM requerían de la modificación de la aplicación exprofeso para adaptarla a un tamaño de memoria reducido. Esto supone probar dos programas distintos para Intel y para ARM, por tanto no aportaría conclusiones y medidas relevantes, no haría uso de las plenas capacidades de ambas máquinas, puesto que se reduciría la cantidad de memoria empleada en la arquitectura Intel. Se recomienda por tanto realizar este mismo estudio con una máquina ARM con más memoria RAM. Es relevante decir que la reserva de memoria de este programa se debería de realizar de forma dinámica, adaptándose a las capacidades de máquina que ejecuta el test.

Dado que únicamente se dispone de la versión secuencial y con Pthreads en Intel se puede decir que este tipo de aplicaciones con mucha comunicación y necesidades de recursos deben ser ejecutadas en una arquitectura x86.

No se puede por tanto comparar las arquitecturas en este test.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1		1		1
	2				1,09
	4				1,13
	8				1,11
Simmedium	1		1		1
	2				1,13
	4				1,21
	8				1,19
Simlarge	1		1		1
	2				1,15
	4				1,23
	8				1,22
Native	1		1		1
	2				1,55
	4				2,14
	8				2,10

Tabla 8 Speed-Up Canneal

Únicamente podemos decir que la versión con 4 hilos el comportamiento es mucho mejor en la arquitectura x86.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1		0,0271		0,0268
	2				0,0247
	4				0,0238
	8				0,0241
Simmedium	1		0,0685		0,0679
	2				0,0601
	4				0,0563
	8				0,0569
Simlarge	1		0,1582		0,1586
	2				0,1384
	4				0,1285
	8				0,1297
Native	1		4,1519		4,1358
	2				2,6631
	4				1,9352
	8				1,9715

Tabla 9 Energía consumida Canneal

Se recomienda el uso de paralelización para este tipo de problemas. Por supuesto se recomienda el uso de una arquitectura Intel x86. Si se dispone posteriormente de una máquina más potente que use ARM se anima a repetir este test. Este test es muy significativo en problemas de minimización y por tanto este tipo de aplicaciones está muy presente en el sector industrial. Se espera a priori que este test no funcionara, dada la poca capacidad de la máquina empleada.

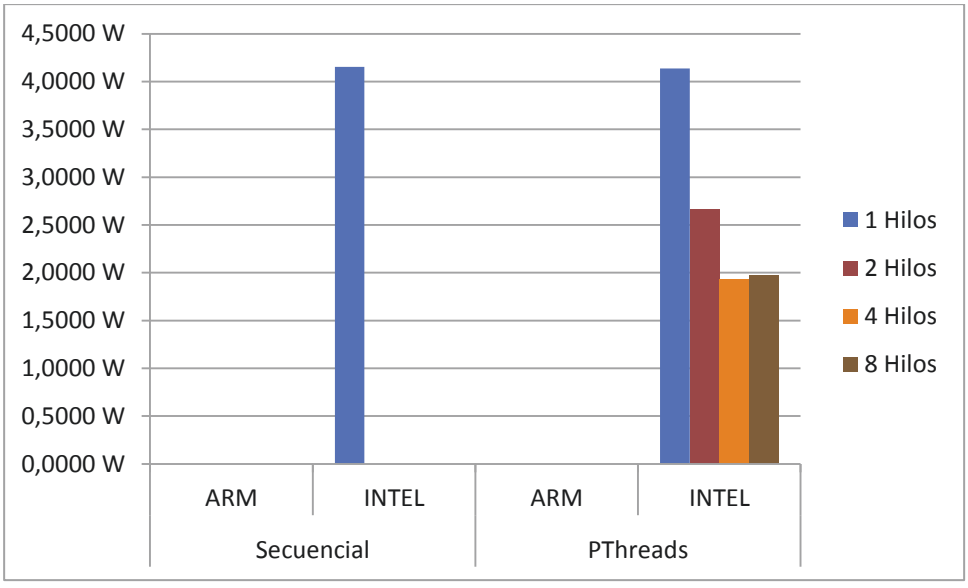

Figura 14 Consumo energético Test Canneal con carga nativa en Intel

3.4.4 TEST DEDUP

Otro *kernel* desarrollado en Princeton. Comprime un flujo de datos con una combinación de compresión global y compresión local de modo que se consiga unas tasas de compresión muy altas. Esta técnica es extremadamente habitual en sistemas de *backup* de todo el mundo. La estrategia seguida para repartir la carga de trabajo es el pipeline con poca carga de trabajo por hilo y con mucha comunicación e intercambio.

Se muestran a continuación las medidas del tiempo para el test en segundos:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1		0,5989		0,6030
	2				0,4819
	4				0,4711
	8				0,5543
Simmedium	1		1,2370		1,2404
	2				0,8937
	4				0,9184
	8				1,1540
Simlarge	1		7,2198		7,1859
	2				5,4543
	4				5,0645
	8				6,2485
Native	1		26,0829		25,3379
	2				20,6482
	4				20,9250
	8				22,8672

Tabla 10 Tiempos Test Dedup

Para este test ha ocurrido lo mismo que en el anterior, requiere una máquina con capacidades mayores dada la gran cantidad de memoria necesaria para generar las estructuras de datos que gestiona. La aplicación dedup también puede ser adaptada para reducir el tamaño de los buffers necesarios. La modificación de la funcionalidad de la aplicación no es el principal objetivo de este estudio, por ello, se recomienda repetir este test en una máquina con capacidades mayores.

No se ha observado ninguna anomalía en el uso de recursos por parte de la arquitectura Intel.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1		1		1
	2				1,25
	4				1,28
	8				1,09
Simmedium	1		1		1
	2				1,39
	4				1,35
	8				1,07
Simlarge	1		1		1
	2				1,32
	4				1,42
	8				1,15
Native	1		1		1
	2				1,23
	4				1,21
	8				1,11

Tabla 11 Speed-Up Dedup

A pesar de solo disponer de la versión con Pthreads en la máquina Intel, comprobamos que el Speed-Up conseguido por la versión paralela no supone más que un 21% de mejora con datos reales y el máximo de hilos disponibles. Se comprueba que la compresión a la cual se somete el flujo de datos no permite una división del trabajo más eficiente.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1		0,5989		0,0159
	2				0,0127
	4				0,0124
	8				0,0146
Simmedium	1		0,0326		0,0327
	2				0,0236
	4				0,0242
	8				0,0305
Simlarge	1		0,1905		0,1896
	2				0,1439
	4				0,1336
	8				0,1649
Native	1		0,6883		0,6686
	2				0,5449
	4				0,5522
	8				0,6034

Tabla 12 Energía consumida Dedup

No se han podido comparar las arquitecturas, por tanto el único comentario a nivel energético es recomendar el uso del API de programación paralela que ofrece la biblioteca de Posix en Intel.

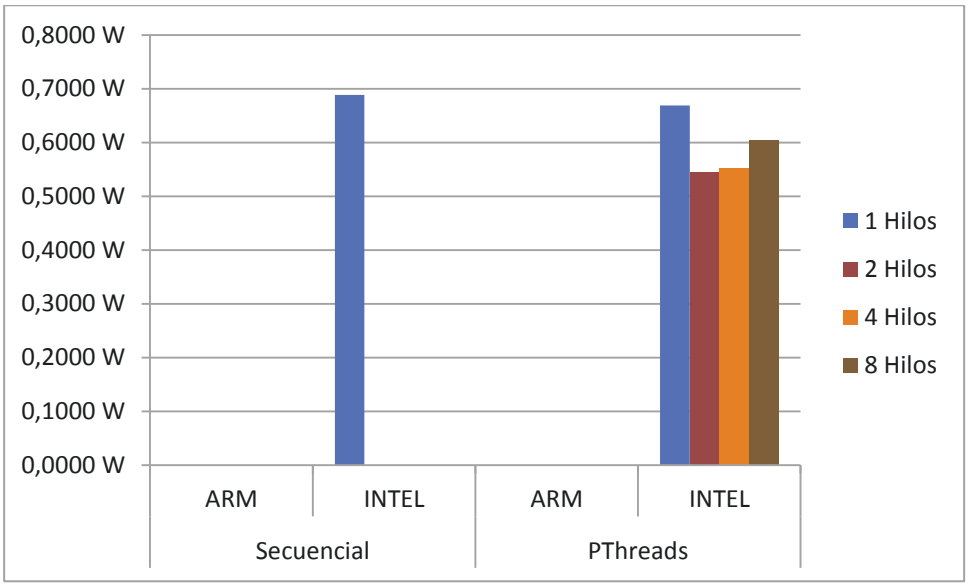

Figura 15 Consumo energético Test Dedup con carga nativa en Intel

3.4.5 TEST FACESIM

Aplicación de modelado desarrollada en colaboración de la Universidad de Stanford e Intel que realiza la simulación de una cara. El modelado se realiza simulando la física que se asocia al rostro. En función de la cantidad de detalles y animaciones la aplicación emplea muchos más recursos. Se reparte la carga paralelamente con grandes cargas por hilo y poca comunicación entre ellos.

Se muestran a continuación las medidas del tiempo para el test en segundos:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	34,8076	4,5910	34,7613	4,6024
	2			26,1450	3,1260
	4			23,5404	2,4431
	8			23,4238	2,4465
Simmedium	1	34,7557	4,5750	34,7873	4,5879
	2			26,1144	3,1268
	4			23,4952	2,4475
	8			23,3996	2,4406
Simlarge	1	34,7377	4,5772	34,8750	4,6053
	2			26,1536	3,1186
	4			23,5263	2,4488
	8			23,4617	2,4418
Native	1	2339,2509	336,2179	2343,0001	338,9139
	2			1548,5359	182,6868
	4			1257,9435	113,6644
	8			1203,4372	112,7222

Tabla 13 Tiempos Test Facesim

A nivel de rendimiento, se cumplen las previsiones de que Intel obtiene los mejores resultados. Cabe mencionar que obtienen resultados muy buenos empleando el API de paralelización Pthreads con 4 hilos. Respecto al resto de eventos del procesador en este tipo de aplicaciones emplean mucho la memoria cache para almacenar los datos parciales de las matrices durante la simulación. Se observa que en ARM se produce una tasa de fallos al referenciar a la cache de un tercio de las veces. En Intel se produce una tasa de solo la mitad. Así mismo se observa que el número de referencias aumenta con el número de hilos empleados en ambas.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1
	2			1,33	1,47
	4			1,48	1,88
	8			1,48	1,88
Simmedium	1	1	1	1	1
	2			1,33	1,47
	4			1,48	1,87
	8			1,49	1,88
Simlarge	1	1	1	1	1
	2			1,33	1,48
	4			1,48	1,88
	8			1,49	1,89
Native	1	1	1	1	1
	2			1,51	1,86
	4			1,86	2,98
	8			1,95	3,01

Tabla 14 Speed-Up Facesim

En esta aplicación observamos que Intel consigue un mejor aprovechamiento de los recursos empleando Pthreads. Estos datos se justifican en el tamaño de la carga por hilo y la poca comunicación, dado que hay pocas zonas en las que haya que esperar por información. Intel consigue resultados mejores dada la frecuencia de cada procesador asociado al hilo.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	0,0967	0,1212	0,0966	0,1215
	2			0,0726	0,0825
	4			0,0654	0,0645
	8			0,0651	0,0646
Simmedium	1	0,0965	0,1207	0,0966	0,1211
	2			0,0725	0,0825
	4			0,0653	0,0646
	8			0,0650	0,0644
Simlarge	1	0,0965	0,1208	0,0969	0,1215
	2			0,0726	0,0823
	4			0,0654	0,0646
	8			0,0652	0,0644
Native	1	6,4979	8,8724	6,5083	8,9436
	2			4,3015	4,8209
	4			3,4943	2,9995
	8			3,3429	2,9746

Tabla 15 Energía consumida Facesim

Destacamos en apartado energético el consumo conseguido por la arquitectura Intel empleando Pthreads puesto que ha conseguido menor consumo que ARM en la versión con 4 hilos. Ofrece por tanto un rendimiento muy superior. Se recomienda por tanto el uso de esta arquitectura para aplicaciones de simulación y renderizado. En caso de no disponer de una versión paralelizada de este tipo de aplicaciones, se puede emplear ARM para renderizado, pero solo en casos en los que no se espere un resultado inmediato.

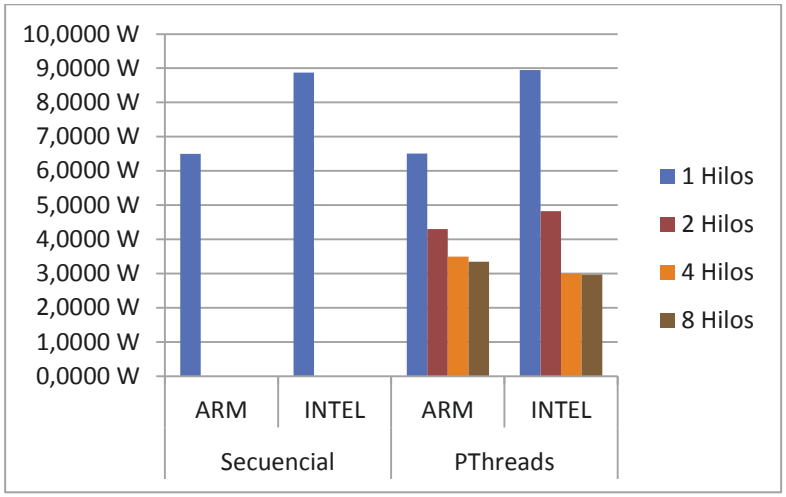

Figura 16 Consumo energético Test Facesim con carga nativa

3.4.6 TEST FERRET

Basado en el kit de herramientas Ferret destinado a buscar contenido similar. Esta herramienta ha sido desarrollada por la universidad de Princeton. Ha sido incluido dado que representa a la próxima generación de motores de búsqueda en archivos que no son textuales, en este caso imágenes y contenido multimedia. Este tipo de operaciones requieren un grado de rendimiento muy alto. Utiliza pipeline como método de paralelización, dividiendo las etapas de la búsqueda entre diferentes hilos. La granularidad es media con alta comunicación entre los hilos.

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	0,6226	0,2515	0,6409	0,2518
	2			0,6084	0,1646
	4			0,5822	0,1535
	8			0,5896	0,1528
Simmedium	1	1,9429	0,6658	1,9543	0,6665
	2			1,9398	0,4241
	4			1,9537	0,4200
	8			1,9487	0,4233
Simlarge	1	4,4943	2,7388	4,5057	2,7353
	2			4,5086	1,8932
	4			4,4909	1,9125
	8			4,4888	1,9489
Native	1		248,3069		246,6534
	2				129,4694
	4				92,3077
	8				92,5087

Tabla 16 Tiempos Test Ferret

En ARM se ha podido obtener únicamente muestras para tamaños de entrada pequeños. Esta prueba se dedica a buscar similitudes entre archivos no de texto, al introducir un gran conjunto de datos, la memoria RAM necesaria aumenta considerablemente. Nos fijamos en que las técnicas de paralelismo apenas han aportado mejora en ARM. En un primer momento se pensó que el test no ejecutaba correctamente. Tras analizar el código de la aplicación se comprobó que con un tamaño pequeño de entrada, se alcanzaban las secciones de paralelismo, pero tenían tan poca carga de trabajo que apenas afectaban al rendimiento. Se puede comprobar que en Intel ocurre lo mismo, siendo la diferencia de pocas décimas de segundo.

Pasados los primeros instantes de la prueba nativa en ARM se produce un fallo a la hora de reservar la memoria. Por tanto se recomienda repetir este test con una máquina más potente.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1
	2			1,05	1,53
	4			1,10	1,64
	8			1,09	1,65
Simmedium	1	1	1	1	1
	2			1,01	1,57
	4			1,00	1,59
	8			1,00	1,57
Simlarge	1	1	1	1	1
	2			1,00	1,44
	4			1,00	1,43
	8			1,00	1,40
Native	1		1		1
	2				1,91
	4				2,67
	8				2,67

Tabla 17 Speed-Up Ferret

Solamente cabe reseñar que el Speed-Up que consigue Intel es muy bueno en este programa tipo pipeline. La división de las etapas y los cálculos se realiza muy eficientemente y la arquitectura Intel consigue una mejora significativa al aplicar el API de Posix Threads.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		PThreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	0,0017	0,0066	0,0018	0,0066
	2			0,0017	0,0043
	4			0,0016	0,0041
	8			0,0016	0,0040
Simmedium	1	0,0054	0,0176	0,0054	0,0176
	2			0,0054	0,0112
	4			0,0054	0,0111
	8			0,0054	0,0112
Simlarge	1	0,0125	0,0723	0,0125	0,0722
	2			0,0125	0,0500
	4			0,0125	0,0505
	8			0,0125	0,0514
Native	1		6,5525		6,5089
	2				3,4166
	4				2,4359
	8				2,4412

Tabla 18 Energía consumida Ferret

Solo se puede hacer a nivel energético la misma observación que a nivel temporal, es necesario realizar este test con una máquina con más recursos.

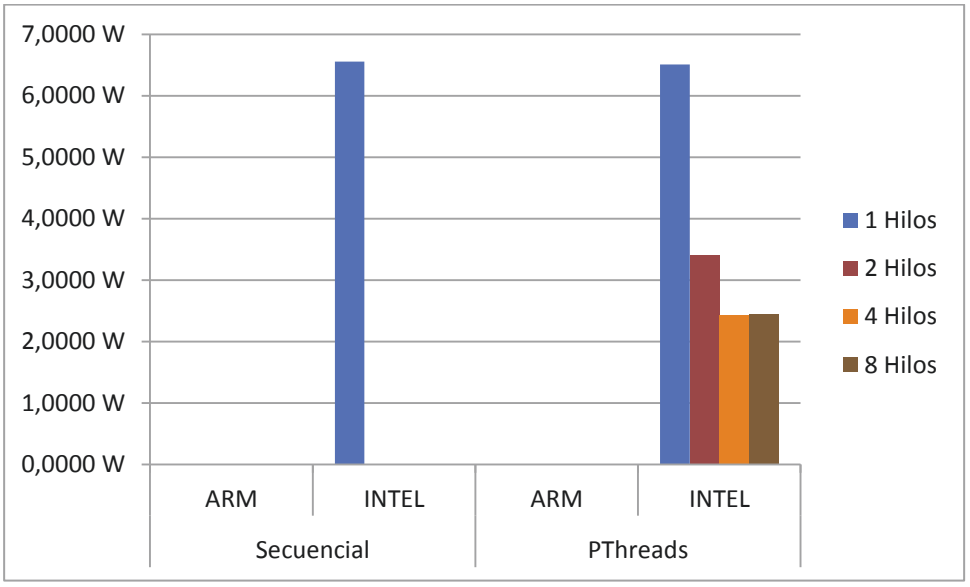

Figura 17 Consumo energético Test Ferret con carga nativa

3.4.7 TEST FLUIDANIMATE

Aplicación Intel empleada para la animación de simulaciones en dinámica de fluidos y partículas. Principalmente incluida en Parsec por su gran utilidad en el campo de la animación de físicas. Actualmente un campo que es pilar fundamental en videojuegos y contenido multimedia. La estrategia de paralelización es dividir las diferentes operaciones entre los hilos, empleando pipeline. No hay mucha comunicación y compartición de datos entre las diferentes operaciones asociadas a los hilos.

Se muestran a continuación las medidas del tiempo para el test en segundos:

Tamaño	Hilos	Secuencial		PThreads		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	1,9339	0,2862	2,2533	0,2853	2,1334	0,2699
	2			1,5109	0,1745	1,3817	0,1635
	4			1,1221	0,1135	1,0334	0,1151
	8			1,1706	0,1330	1,1069	0,1251
Simmedium	1	4,4782	0,5510	5,2578	0,5526	5,2612	0,5296
	2			3,4915	0,3550	3,4199	0,3430
	4			2,4814	0,2413	2,4878	0,2303
	8			2,5999	0,2836	2,5820	0,2574
Simlarge	1	13,0472	1,6101	15,5417	1,6213	15,4755	1,5860
	2			9,8128	0,9994	9,7459	0,9828
	4			7,3206	0,6681	7,2268	0,6627
	8			7,6592	0,7353	7,5404	0,7232
Native	1	1600,6975	226,7678	2006,2743	228,2680	2008,0874	228,2314
	2			1143,2307	126,5924	1055,4970	125,0892
	4			829,7431	75,0309	722,1473	74,5801
	8			827,7315	81,8669	742,6908	81,0917

Tabla 19 Tiempos Test Fluidanimate

Se puede observar que Intel TBB da mejores resultados que Pthreads en ambas arquitecturas. Achacamos este fenómeno a la estrategia que ofrece TBB implementando la técnica del robo de tareas en este programa concreto. Dado que para la simulación de los fluidos se debe de calcular la densidad para todas las partículas, cuando un hilo termina el cálculo, puede encargarse de ayudar. En este caso para en el cálculo de otra partícula. Lo mismo ocurre con el cálculo de las fuerzas y las colisiones de las partículas.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		PThreads		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1	1	1
	2			1,49	1,63	1,54	1,65
	4			2,01	2,51	2,06	2,34
	8			1,92	2,15	1,93	2,16
Simmedium	1	1	1	1	1	1	1
	2			1,51	1,56	1,54	1,54
	4			2,12	2,29	2,11	2,30
	8			2,02	1,95	2,04	2,06
Simlarge	1	1	1	1	1	1	1
	2			1,58	1,62	1,59	1,61
	4			2,12	2,43	2,14	2,39
	8			2,03	2,20	2,05	2,19
Native	1	1	1	1	1	1	1
	2			1,75	1,80	1,90	1,82
	4			2,42	3,04	2,78	3,06
	8			2,42	2,79	2,70	2,81

Tabla 20 Speed-Up Fluidanimate

En este test se comprueba que Intel TBB ha conseguido mejores resultados en lo referente al Speed-Up para ambas arquitecturas. Como se puede observar, Intel aprovecha mejor las técnicas de robo de tareas. Se justifica la mejora en Intel debida a la poca comunicación que hay entre hilos en esta aplicación. Cuando una tarea termina en Intel, usualmente más rápido, puede destinar hilos a ayudar a otras tareas más rápidamente que en ARM.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		PThreads		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	0,0054	0,0076	0,0063	0,0075	0,0059	0,0071
	2			0,0042	0,0046	0,0038	0,0043
	4			0,0031	0,0030	0,0029	0,0030
	8			0,0033	0,0035	0,0031	0,0033
Simmedium	1	0,0124	0,0145	0,0146	0,0146	0,0146	0,0140
	2			0,0097	0,0094	0,0095	0,0091
	4			0,0069	0,0064	0,0069	0,0061
	8			0,0072	0,0075	0,0072	0,0068
Simlarge	1	0,0362	0,0425	0,0432	0,0428	0,0430	0,0419
	2			0,0273	0,0264	0,0271	0,0259
	4			0,0203	0,0176	0,0201	0,0175
	8			0,0213	0,0194	0,0209	0,0191
Native	1	4,4464	5,9842	5,5730	6,0237	5,5780	6,0228
	2			3,1756	3,3406	2,9319	3,3010
	4			2,3048	1,9800	2,0060	1,9681
	8			2,1068	2,1604	2,0630	2,1399

Tabla 21 Energía consumida Fluidanimate

En esta prueba, Intel obtiene muy buenos resultados a nivel energético. Se asemeja mucho a ARM con un rendimiento muy superior. Es por tanto para este test la arquitectura recomendada a la hora de realizar animaciones en tiempo real. ARM puede ser empleado para renderizado pero con las características actuales de la máquina, no tiene sentido su uso en animación dado el tiempo que tarda en procesar la información. En cambio, si se puede usar como alternativa al escalar el problema, con varias máquinas, contanto con más hilos, en este tipo de programas se puede repartir la carga dada la granularidad fina.

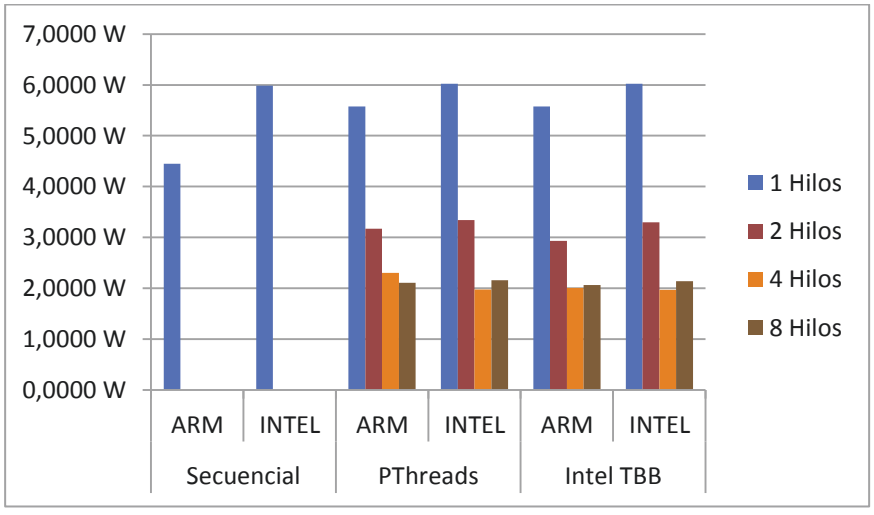

Figura 18 Consumo energético Test Fluidanimate con carga nativa

3.4.8 TEST FREQMINE

Implementa un método de minería de datos basado en arrays empleando FP-*growth* (Patrones de crecimiento frecuente) para Minería de conjuntos de elementos frecuentes. Es un buen ejemplo de aplicación de minería de datos compleja. Se reparte la carga de trabajo entre los hilos con un modelo de cálculo de datos paralelos. Se produce mucha comunicación de los hilos en la etapa de consolidación de las búsquedas.

Tamaño	Hilos	Secuencial		OpenMP	
		ARM	INTEL	ARM	INTEL
Simsmall	1	2,5750	0,4237	2,5722	0,4289
	2			1,8554	0,2689
	4			1,5241	0,1917
	8			1,5433	0,1969
Simmedium	1	8,2783	1,5051	8,3603	1,5251
	2			5,2332	0,8707
	4			3,6639	0,5411
	8			3,8913	0,5981
Simlarge	1	27,1885	0,8159	27,6216	0,8462
	2			15,7444	0,6183
	4			11,7358	0,5092
	8			12,0581	0,5224
Native	1	1631,9930	404,7907	1973,8009	419,2728
	2			1198,9922	214,1558
	4				114,1305
	8				112,8470

Tabla 22 Tiempos Test Freqmine

En esta prueba se puede comprobar que el procesador Intel obtiene un resultado muy superior en términos de rendimiento. Por tanto, podemos concluir que si el tiempo es un factor relevante, Intel es la arquitectura que mejor se comporta en problemas de minería de datos. Además, se recomienda encarecidamente el uso de técnicas de paralelismo. Con máquinas con más procesadores se pueden obtener mejores resultados dada la granularidad media del problema. Únicamente se puede encontrar algún problema al escalar el problema dada la alta comunicación que emplea este programa.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		OpenMP	
		ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1
	2			1,39	1,60
	4			1,69	2,24
	8			1,67	2,18
Simmedium	1	1	1	1	1
	2			1,60	1,75
	4			2,28	2,82
	8			2,15	2,55
Simlarge	1	1	1	1	1
	2			1,75	1,37
	4			2,35	1,66
	8			2,29	1,62
Native	1	1	1	1	1
	2			1,65	1,96
	4				3,67
	8				3,72

Tabla 23 Speed-Up Freqmine

Pese a ello, las pruebas demuestran que Intel consigue sacar mejor partido al API OpenMP para cargas grandes. Los tiempos empleados en comunicar los hilos puede afectar notablemente al rendimiento, produciéndose en muchos casos esperas por datos en ARM. Para cargas pequeñas, esta arquitectura consigue mejoras significativas en Speed-Up. Para cargas grandes solo se pueden aportar datos de mejora con 2 hilos, donde Intel se muestra superior respecto a la gestión de los hilos. Al tardar menos en calcular por hilo, consigue menor tiempo de espera al consolidar los datos por hilo.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		OpenMP	
		ARM	INTEL	ARM	INTEL
Simsmall	1	0,0072	0,0112	0,0071	0,0113
	2			0,0052	0,0071
	4			0,0042	0,0051
	8			0,0043	0,0052
Simmedium	1	0,0230	0,0397	0,0232	0,0402
	2			0,0145	0,0230
	4			0,0102	0,0143
	8			0,0108	0,0158
Simlarge	1	0,0755	0,0215	0,0767	0,0223
	2			0,0437	0,0163
	4			0,0326	0,0134
	8			0,0335	0,0138
Native	1	4,5333	10,6820	5,4828	11,0641
	2			3,3305	5,6513
	4			1,0764	3,0118
	8			0,3420	2,9779

Tabla 24 Energía consumida Freqmine

A la vista de los datos relativos a nivel energético se recomienda el uso de la arquitectura ARM para tareas de minería de datos que no requieran de resultados inmediatos, se puede emplear un clúster de máquinas ARM para realizar los cálculos de las diferentes secciones de paralelismo. Si la tarea no implica tener un rendimiento alto, el ahorro energético que se produce empleando ARM es de prácticamente la mitad de potencia.

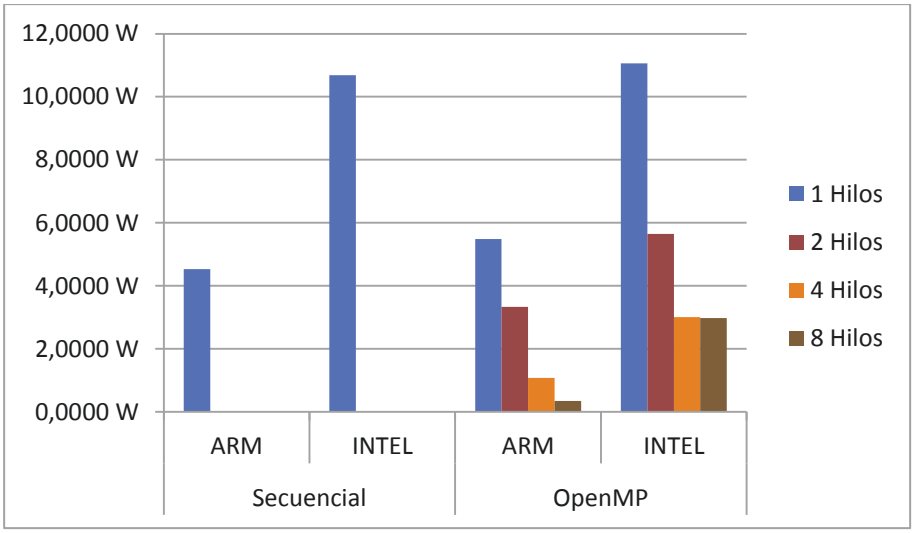

Figura 19 Consumo energético Test Freqmine con carga nativa

3.4.9 TEST RAYTRACE

Desarrollada inicialmente por Intel este método de animación en tiempo real empleado para juegos prima la velocidad por encima del realismo y es dependiente de la resolución que se pretenda conseguir. La complejidad del algoritmo por tanto es dependiente de dicha resolución. Es interesante poder estudiar este programa en arquitecturas usadas actualmente para móviles y tabletas que cada vez demandan mejores capacidades gráficas. La carga al paralelizar sigue el modelo de datos paralelos, repartiendo la carga de las operaciones de cálculo con granularidad media. La compartición de datos y la comunicación entre los hilos también es media.

Se muestran a continuación las medidas del tiempo para el test en segundos:

Tamaño	Hilos	Secuencial		Pthreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	33,0972	4,6022	33,0787	4,5545
	2			31,8621	4,4713
	4			31,9470	4,4540
	8			31,8364	4,4272
Simmedium	1	34,6740	4,9351	34,6840	4,8924
	2			31,9338	4,6332
	4			31,9052	4,5345
	8			31,9143	4,5209
Simlarge	1	38,2368	5,7987	38,2525	5,7522
	2			31,8643	5,1167
	4			31,8595	4,7725
	8			31,9384	4,7678
Native	1		200,2802		199,6180
	2				129,1035
	4				94,5499
	8				94,0344

Tabla 25 Tiempos Test Raytrace

Pese a que para cargas pequeñas se ha podido simular en la arquitectura ARM, no se ha podido conseguir una muestra para una carga real. Por tanto en este test solo se pueden extraer conclusiones para cargas pequeñas. No se recomienda el uso de este tipo de programas de simulación en la arquitectura ARM. Esta recomendación se extiende para todas las aplicaciones de simulación en tiempo real. Era de esperar este tipo de comportamiento en aplicaciones que necesitan almacenar gran cantidad de datos en memoria.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		Pthreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1
	2			1,04	1,02
	4			1,04	1,03
	8			1,04	1,02
Simmedium	1	1	1	1	1
	2			1,09	1,06
	4			1,09	1,08
	8			1,09	1,08
Simlarge	1	1	1	1	1
	2			1,20	1,12
	4			1,20	1,21
	8			1,20	1,21
Native	1		1		1
	2				1,55
	4				2,11
	8				2,12

Tabla 26 Speed-Up Raytrace

Los datos que se muestran para la mejora de rendimiento al paralelizar con Pthreads no resultan relevantes para ARM, puesto que solo se consigue resultado para cargas de simulación. En lo referente a la carga nativa, Intel consigue un rendimiento óptimo empleando los cuatro procesadores con Pthreads.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		OpenMP	
		ARM	INTEL	ARM	INTEL
Simsmall	1	0,0072	0,0112	0,0919	0,1202
	2			0,0885	0,1180
	4			0,0887	0,1175
	8			0,0884	0,1168
Simmedium	1	0,0963	0,1302	0,0963	0,1291
	2			0,0887	0,1223
	4			0,0886	0,1197
	8			0,0887	0,1193
Simlarge	1	0,1062	0,1530	0,1063	0,1518
	2			0,0885	0,1350
	4			0,0885	0,1259
	8			0,0887	0,1258
Native	1		5,2852		5,2677
	2				3,4069
	4				2,4951
	8				2,4815

Tabla 27 Energía consumida Raytrace

A nivel de consumo energético, no merece la pena reseñar nada especial. Intel para cargas pequeñas apenas consume un poco más que ARM. Por tanto en este sentido se recomienda el uso de Intel parar simulación en tiempo real.

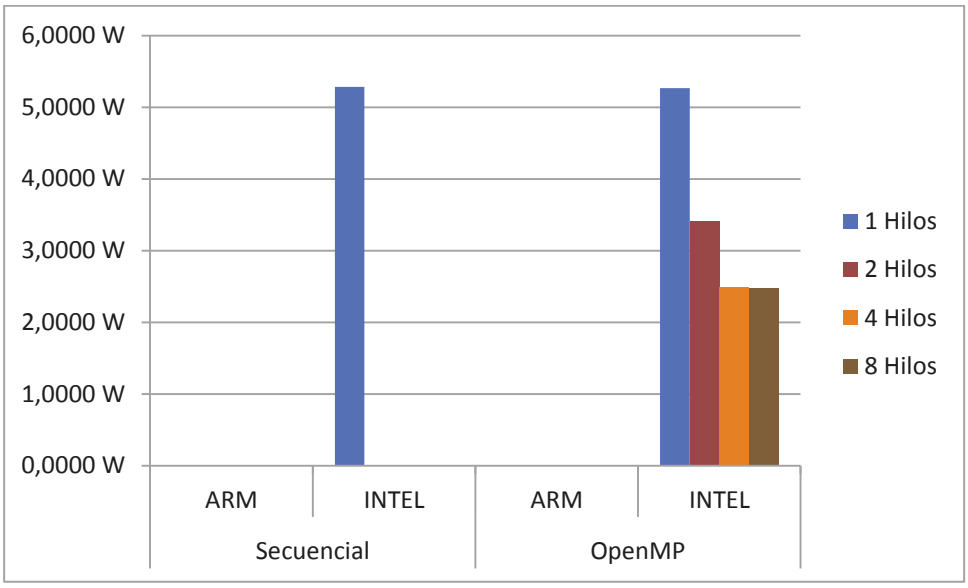

Figura 20 Consumo energético Test Raytrace con carga nativa

3.4.10 TEST STREAMCLUSTER

Kernel que trata de resolver el problema del *clustering* online. Se trata de un algoritmo de minería de datos incluido por componer un problema de minería de datos con flujos de datos constantes. Sigue un modelo de paralelización dividiendo el trabajo de igual forma entre los distintos hilos, con poca comunicación e intercambio de datos entre ellos.

Se muestran a continuación las medidas del tiempo para el test en segundos:

Tamaño	Hilos	Secuencial		PThreads		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	1,2806	0,1871	1,2533	0,1880	1,2127	0,1907
	2			0,7406	0,1077	0,7173	0,1160
	4			0,5234	0,1291	0,4622	0,0805
	8			9,8312	2,2617	0,5111	0,0866
Simmedium	1	7,8185	0,7449	7,8259	0,7423	7,8029	0,7286
	2			5,0183	0,4010	4,8850	0,3991
	4			4,6423	0,3031	4,4822	0,2390
	8			15,2020	2,7416	4,7669	0,2587
Simlarge	1	30,5445	5,5550	30,9889	5,5250	30,9864	5,5731
	2			22,9428	2,9522	21,5915	2,9811
	4			22,0939	1,8252	21,9270	1,7377
	8			34,6939	4,9537	23,1819	1,7740
Native	1	1777,8866	352,8654	1780,7104	353,4721	1807,9762	343,7344
	2			1221,9126	190,8400	1211,5610	183,6106
	4			1204,8925	111,4271	981,9907	105,1876
	8			1269,8022	128,2629	991,9907	107,3885

Tabla 28 Tiempos Test Streamcluster

En lo referente a esta prueba cabe reseñar que ARM no ha podido gestionar correctamente la versión con 8 hilos en TBB. Intel sigue siendo la arquitectura más recomendada para obtener un rendimiento óptimo. El uso de TBB se recomienda en este tipo de aplicaciones de minería de datos.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		PThreads		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1	1	1
	2			1,69	1,75	1,69	1,64
	4			2,39	1,46	2,62	2,37
	8			0,13	0,08	2,37	2,20
Simmedium	1	1	1	1	1	1	1
	2			1,56	1,85	1,60	1,83
	4			1,69	2,45	1,74	3,05
	8			0,51	0,27	1,64	2,82
Simlarge	1	1	1	1	1	1	1
	2			1,35	1,87	1,44	1,87
	4			1,40	3,03	1,41	3,21
	8			0,89	1,12	1,34	3,14
Native	1	1	1	1	1	1	1
	2			1,46	1,85	1,49	1,87
	4			1,48	3,17	1,84	3,27
	8			1,40	2,76	1,80	3,20

Tabla 29 Speed-Up Streamcluster

Intel es la arquitectura que claramente se beneficia mejor de los APIs de programación paralela. Consigue prácticamente el doble de rendimiento con el máximo de hilos configurado y con cargas grandes. Por otro lado ARM mejora pero no consigue beneficiarse eficientemente del API Pthreads, no obtiene mejoras de establecer diferentes hilos. Dada la gran carga por hilo, los tiempos de espera se disparan en ARM al solo tener que consolidar por los datos del minado.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		PThreads		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	0,0036	0,0049	0,0035	0,0050	0,0034	0,0050
	2			0,0021	0,0028	0,0020	0,0031
	4			0,0015	0,0034	0,0013	0,0021
	8			0,0273	0,0597	0,0014	0,0023
Simmedium	1	0,0217	0,0197	0,0217	0,0196	0,0217	0,0192
	2			0,0139	0,0106	0,0136	0,0105
	4			0,0129	0,0080	0,0125	0,0063
	8			0,0422	0,0723	0,0132	0,0068
Simlarge	1	0,0848	0,1466	0,0861	0,1458	0,0861	0,1471
	2			0,0637	0,0779	0,0600	0,0787
	4			0,0614	0,0482	0,0609	0,0459
	8			0,0964	0,1307	0,0644	0,0468
Native	1	4,9386	9,3117	4,9464	9,3277	5,0222	9,0708
	2			3,3942	5,0361	3,3654	4,8453
	4			3,3469	2,9404	2,7278	2,7758
	8			3,5272	3,3847	0,0001	2,8339

Tabla 30 Energía consumida Streamcluster

Energéticamente ARM obtiene ventaja reduciendo prácticamente a la mitad el consumo demandado. Por tanto se vuelve a recomendar el uso de esta arquitectura en situaciones en las que el rendimiento no sea primordial.

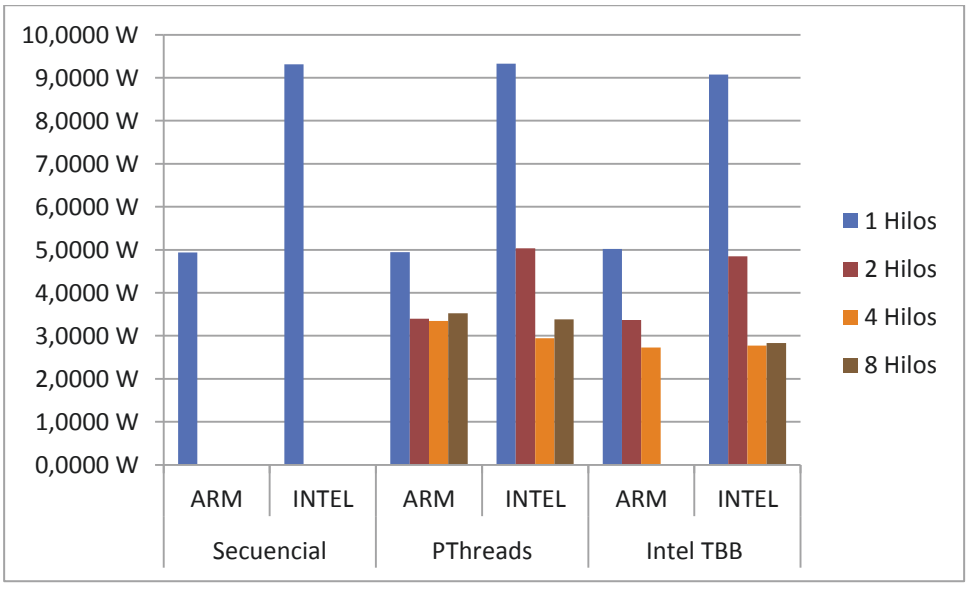

Figura 21 Consumo energético Test Streamcluster con carga nativa

3.4.11 TEST SWAPTIONS

Programa que implementa el algoritmo HJM (*Hearth-Jarrow-Morton*) para el cálculo del precio de activos conocidos como swaptions. Usa el método aproximación Monte Carlo para calcular dichos precios. Paraleliza el cálculo de los precios repartiendo la carga entre los diferentes hilos, con poca comunicación e intercambio entre ellos.

Se muestran a continuación las medidas del tiempo para el test en segundos:

Tamaño	Hilos	Secuencial		PThreads		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	0,9273	0,1403	0,9307	0,1368	0,9297	0,1435
	2			0,5339	0,0763	0,5156	0,0790
	4			0,3127	0,0457	0,3345	0,0515
	8			0,3245	0,0496	0,3523	0,0554
Simmedium	1	3,2687	0,5175	3,2584	0,5184	3,3627	0,5430
	2			1,6870	0,2713	1,7248	0,2850
	4			0,8953	0,1508	0,9552	0,1597
	8			0,9053	0,1579	0,9972	0,1720
Simlarge	1	12,6194	2,0380	12,6286	2,0401	13,0806	2,1322
	2			6,8738	1,0549	6,6665	1,1081
	4			4,4926	0,5682	4,8017	0,5945
	8			4,5745	0,5792	5,1373	0,6205
Native	1	1248,7777	200,2125	1248,7797	199,3301	1296,5276	210,0585
	2			791,2774	103,2558	813,3956	109,1204
	4			516,2413	54,1968	521,0351	57,8405
	8			518,2058	54,3270	533,7257	57,6975

Tabla 31 Tiempos Test Swaptions

Con los resultados obtenidos se consigue un rendimiento óptimo empleando Pthreads, este comportamiento puede deberse al bajo grado de compartición y la granularidad gruesa del problema. A priori se pensaba que dada la caracterización del problema TBB funcionaría mejor, pero tras realizar el test, se ha comprobado que la versión paralelizada con Pthreads está mejor gestionada que la versión con TBB. En este caso la gestión definida por el usuario de los hilos ha funcionado mejor que la gestión automática de TBB.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		PThreads		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1	1	1
	2			2,98	2,99	2,78	2,79
	4			2,87	2,76	2,64	2,59
	8			1,74	1,79	1,80	1,82
Simmedium	1	1	1	1	1	1	1
	2			3,64	3,44	3,52	3,40
	4			3,60	3,28	3,37	3,16
	8			1,93	1,91	1,95	1,91
Simlarge	1	1	1	1	1	1	1
	2			1,84	1,93	1,96	1,92
	4			2,81	3,59	2,72	3,59
	8			2,76	3,52	2,55	3,44
Native	1	1	1	1	1	1	1
	2			1,58	1,93	1,59	1,93
	4			2,42	3,68	2,49	3,63
	8			2,41	3,67	2,43	3,64

Tabla 32 Speed-Up Swaptions

Para este Test se considera que Intel aprovecha mejor las bibliotecas de paralelismo, consiguiendo la máxima mejora con 4 hilos para ambas. Por contrapartida, ARM consigue obtener un resultado bueno para la carga de simulación media, pero no resulta muy relevante, dado que este tipo de problemas de cálculo de valores financieros suelen requerir grandes cargas de trabajo.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		PThreads		Intel TBB	
		ARM	INTEL	ARM	INTEL	ARM	INTEL
Simsmall	1	0,0026	0,0037	0,0026	0,0036	0,0026	0,0038
	2			0,0015	0,0020	0,0014	0,0021
	4			0,0009	0,0012	0,0009	0,0014
	8			0,0009	0,0013	0,0010	0,0015
Simmedium	1	0,0091	0,0137	0,0091	0,0137	0,0093	0,0143
	2			0,0047	0,0072	0,0048	0,0075
	4			0,0025	0,0040	0,0027	0,0042
	8			0,0025	0,0042	0,0028	0,0045
Simlarge	1	0,0351	0,0538	0,0351	0,0538	0,0363	0,0563
	2			0,0191	0,0278	0,0185	0,0292
	4			0,0125	0,0150	0,0133	0,0157
	8			0,0127	0,0153	0,0143	0,0164
Native	1	3,4688	5,2834	3,4688	5,2601	3,6015	5,5432
	2			2,1980	2,7248	2,2594	2,8796
	4			1,4340	1,4302	1,4473	1,5263
	8			1,4395	1,4336	1,4826	1,5226

Tabla 33 Energía consumida Swaptions

ARM obtiene mejores resultados en el apartado energético, como se esperaba, este tipo de arquitecturas puede ser empleado para análisis financiero. Como para el resto de pruebas en las que ARM obtiene mejores resultados energéticos, se recomienda emplear la arquitectura siempre que el tiempo no sea relevante, por ejemplo para el cálculo de activos en archivos históricos, estudios estadísticos de tendencias en el mercado o previsión de desastres financieros.

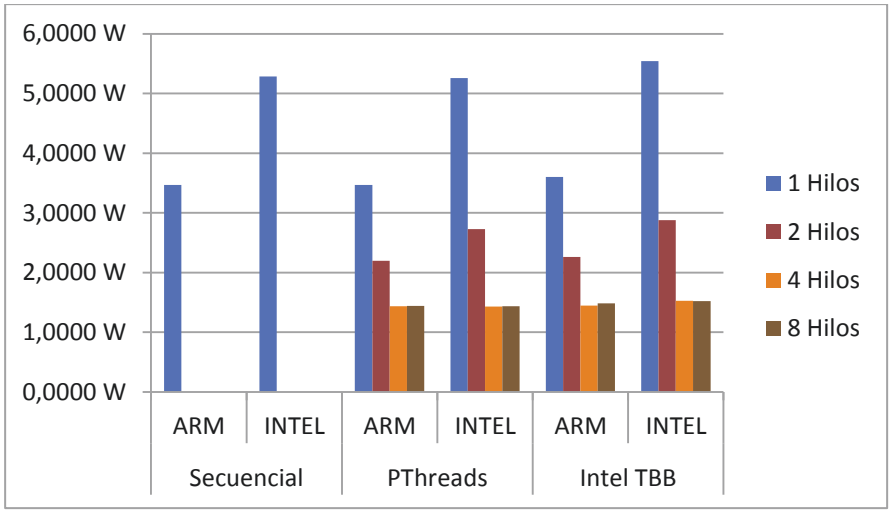

Figura 22 Consumo energético Test Streamcluster con carga nativa

3.4.12 TEST VIPS

Aplicación desarrollada con el sistema de procesamiento de imágenes *VASARI* para procesamiento de imágenes de gran tamaño. El algoritmo emplea transformaciones lineales afines tales y convoluciones, muy comunes en el ámbito del procesamiento de imágenes. Sigue un modelo de paralelización de datos dividiendo en secciones las matrices que maneja y asociando la carga a los hilos. Hay poca comunicación entre los hilos y más intercambio de información para consolidar los resultados de los cálculos parciales.

Se muestran a continuación las medidas del tiempo para el test en segundos:

Tamaño	Hilos	Secuencial		Pthreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	4,4927	0,5512	4,5205	0,5457
	2			2,9953	0,2832
	4			2,3335	0,1828
	8			2,3381	0,1736
Simmedium	1	12,5904	1,5851	12,6152	1,5950
	2			7,8448	0,7973
	4			6,7022	0,4727
	8			6,5237	0,4438
Simlarge	1	30,6230	4,3869	30,1841	4,4037
	2			19,1537	2,1825
	4			14,9386	1,2382
	8			15,4652	1,1827
Native	1	636,8556	101,1364	635,8999	102,6491
	2			349,3314	54,6108
	4			267,1554	31,4811
	8			266,2996	28,8571

Tabla 34 Tiempos Test VIPS

Esta aplicación de procesamiento de imágenes obtiene resultados muy buenos en la arquitectura Intel. En comparación con ARM el rendimiento para una aplicación que se presenta de cara al usuario es claramente superior. Pese a este dato, ARM puede ser empleado en realizar las transformaciones afines en bibliotecas de Imágenes de gran tamaño. Respecto a las bibliotecas de paralelización, solo se dispone de datos con Posix Pthreads por tanto solo se puede recomendar el uso de este API.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		Pthreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1
	2			1,51	1,93
	4			1,94	2,99
	8			1,93	3,14
Simmedium	1	1	1	1	1
	2			1,61	2,00
	4			1,88	3,37
	8			1,93	3,59
Simlarge	1	1	1	1	1
	2			1,58	2,02
	4			2,02	3,56
	8			1,95	3,72
Native	1	1	1	1	1
	2			1,82	1,88
	4			2,38	3,26
	8			2,39	3,56

Tabla 35 Speed-Up VIPS

Para el test de se reportan mejores datos de aprovechamiento de bibliotecas paralelas en Intel, consiguiendo resultados óptimos con la máxima configuración de hilos y Pthreads. Si comprobamos los resultados de ARM, también se consigue el máximo aumento de rendimiento con la configuración máxima. Como se puede comprobar, las esperas a la hora de consolidar la información producen un descenso en el Speed-Up en ARM.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		Pthreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	0,0125	0,0145	0,0126	0,0144
	2			0,0083	0,0075
	4			0,0065	0,0048
	8			0,0065	0,0046
Simmedium	1	0,0350	0,0418	0,0350	0,0421
	2			0,0218	0,0210
	4			0,0186	0,0125
	8			0,0181	0,0117
Simlarge	1	0,0851	0,1158	0,0838	0,1162
	2			0,0532	0,0576
	4			0,0415	0,0327
	8			0,0430	0,0312
Native	1	1,7690	2,6689	1,7664	2,7088
	2			0,9704	1,4411
	4			0,7421	0,8308
	8			0,7397	0,7615

Tabla 36 Energía consumida VIPS

Se cumple nuevamente la teoría de que ARM puede resultar una arquitectura útil a la hora de gestionar imágenes y realizar las transformaciones que se requieran. Como norma, se remienda el uso de la arquitectura en operaciones no prioritarias, en este caso, un buen uso de la arquitectura puede ser el cambio de formato de grandes cantidades de archivos, aplicando transformaciones reales y convoluciones tal y como se realizan en esta prueba.

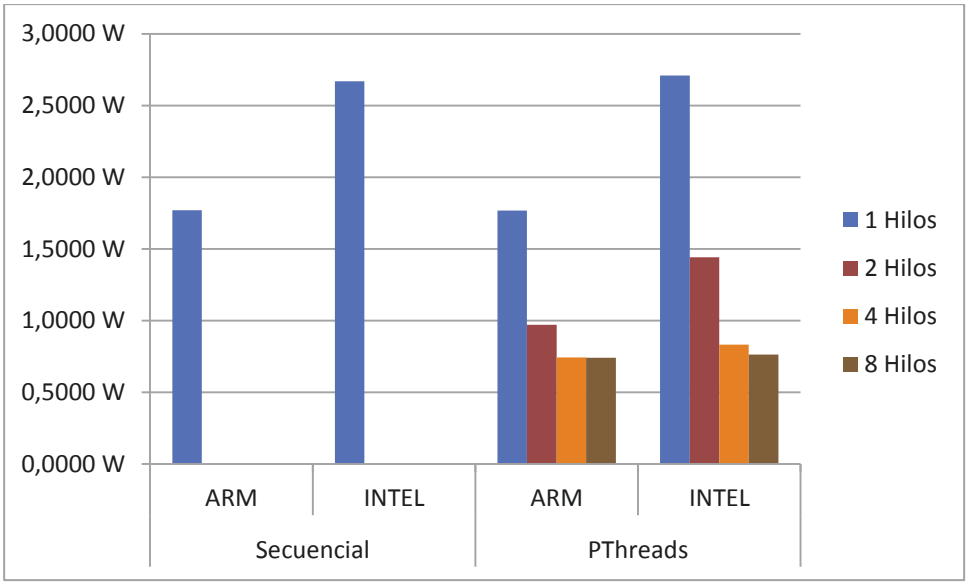

Figura 23 Consumo energético Test VIPS con carga nativa

3.4.13 TEST X264

Codificador de vídeo empleando la codificación *H.264/AVC* de compresión con pérdida de flujos de vídeo (streaming) y parte del estándar ISO de codificación de archivos *MPEG-4* en alta definición. Interesante puesto que la codificación y compresión de video es uno de los procesos más costosos que se conocen en el dominio del contenido multimedia. Para las versiones paralelas se ejecutan en pipeline, siguiendo las etapas de la codificación/compresión/decodificación. La comunicación es por tanto muy alta, así como el intercambio de información entre las etapas.

Se muestran a continuación las medidas del tiempo para el test en segundos:

Tamaño	Hilos	Secuencial		Pthreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	3,1701	0,1436	3,1816	0,1397
	2			2,0500	0,0946
	4			1,6383	0,0758
	8			1,5236	0,0649
Simmedium	1	19,4723	0,7662	19,5047	0,7673
	2			11,3158	0,4294
	4			8,5631	0,2637
	8			8,3504	0,2524
Simlarge	1	62,7793	2,3820	62,7271	2,3702
	2			39,2655	1,2895
	4			27,8167	0,7684
	8			27,3230	0,7444
Native	1	2218,1346	89,4716	2212,0369	82,7348
	2			1312,0864	44,4606
	4			912,8512	26,4646
	8			904,1489	27,0350

Tabla 37 Tiempos Test x264

Se puede comprobar a simple vista que los test sobre la arquitectura ARM han sido muy inferiores respecto a Intel. Con tiempos 25 veces superiores, las operaciones de codificación y decodificación de archivos de vídeo no están recomendadas con esta arquitectura. Este aumento significativo del tiempo se cree que deriva de la granularidad del problema y de la alta compartición de datos necesaria. Al tener que esperar por grandes cargas de trabajo de los hilos, el tiempo que se pasa esperando se acumula muy rápidamente. Este dato unido a que la comunicación entre hilos es alta deriva en un problema mayor para arquitecturas con prestaciones bajas. Las condiciones de carrera y en aplicaciones tipo pipeline son devastadoras para los resultados, puesto que frenan todo el proceso con las esperas.

Se presentan a continuación los cálculos relativos al Speed-Up para las versiones paralelas:

Tamaño	Hilos	Secuencial		Pthreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	1	1	1	1
	2			1,55	1,48
	4			1,94	1,84
	8			2,09	2,15
Simmedium	1	1	1	1	1
	2			1,72	1,79
	4			2,28	2,91
	8			2,34	3,04
Simlarge	1	1	1	1	1
	2			1,60	1,84
	4			2,26	3,08
	8			2,30	3,18
Native	1	1	1	1	1
	2			1,69	1,86
	4			2,42	3,13
	8			2,45	3,06

Tabla 38 Speed-Up x264

En este test se comprueba que nuevamente Intel realiza un uso más eficiente del API Posix Pthreads con configuración máxima. Por otro lado, esta aplicación requiere de un rendimiento alto, puesto que el proceso de compresión de datos para streaming requiere rendimiento alto, dado los tamaños que los archivos de vídeo. En este caso ARM aprovecha realmente las capacidades de los hilos, pero nuevamente los tiempos de a la espera de datos entre las etapas de compresión reducen la mejora.

Se presentan a continuación las medidas del consumo de energía en vatios:

Tamaño	Hilos	Secuencial		Pthreads	
		ARM	INTEL	ARM	INTEL
Simsmall	1	0,0088	0,0038	0,0088	0,0037
	2			0,0057	0,0025
	4			0,0046	0,0020
	8			0,0042	0,0017
Simmedium	1	0,0541	0,0202	0,0542	0,0202
	2			0,0314	0,0113
	4			0,0238	0,0070
	8			0,0232	0,0067
Simlarge	1	0,1744	0,0629	0,1742	0,0625
	2			0,1091	0,0340
	4			0,0773	0,0203
	8			0,0759	0,0196
Native	1	6,1615	2,3611	6,1445	2,1833
	2			3,6447	1,1733
	4			2,5357	0,6984
	8			2,5115	0,7134

Tabla 39 Energía consumida x264

Dado el mal rendimiento obtenido por la máquina ARM no se recomienda su uso para cargas de trabajo grandes. En cualquier caso el rendimiento de un programa de decodificación de video en directo requiere de un rendimiento alto. Solo podemos decir por tanto que Intel es la opción más ventajosa en este tipo de problemas.

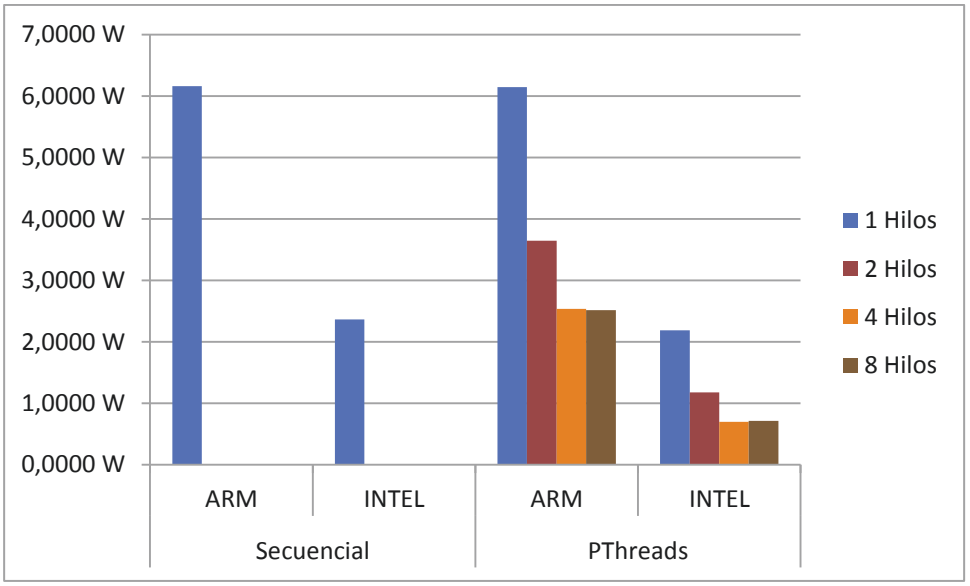

Figura 24 Consumo energético Test x264 con carga nativa

3.4.14 OBSERVACIONES GENERALES DE LAS PRUEBAS

En esta sección se aportan datos referentes al uso de recursos durante las pruebas de un modo general, principalmente a las medidas tomadas por Perf en ambas arquitecturas. Esto ha servido para comprobar que todo ha funcionado según lo esperado y que los recursos se han usado como se esperaba.

A continuación comentamos algunas observaciones relativas al uso de recursos:

- Ciclos: ARM ha empleado un 33% más de ciclos para llevar a cabo las pruebas. Esta estadística cuadra con la filosofía de los juegos de instrucciones y se esperaba.
- Instrucciones: ARM ha ejecutado un 10% más de instrucciones.
- Cachés: ARM ha referenciado más a la cache, obteniendo un 2,71% de fallos, Intel únicamente el 1,90%.
- Saltos: Pese a que Intel ha realizado más saltos, su porcentaje de fallos general ha sido del 2,28% mientras que ARM ha fallado el 12,82% de las veces.
- Operaciones *branch*: ARM ha generado menos eventos *brach*, pero ha fallado el 7,08% de las veces, Intel solo 2,28% de fallos en este tipo de operaciones.
- Respecto al tiempo, las pruebas en Intel han requerido un total de 4 horas y media. Para ARM el tiempo ha sido de 25 horas.
- En total, Intel ha consumido un total de 418 vatios. ARM 281 vatios.

No se han observado anomalías en ninguna de las pruebas superadas, repitiéndose el mismo comportamiento, en líneas generales, para todas ellas y obteniendo el mismo resultado de salida para ambas arquitecturas[3].

[3] Solamente se exceptúan aquellos resultados que incluyen operaciones en punto flotante, dado que los resultados son muy parecidos, pero no exactamente iguales.

3.5 CONCLUSIONES Y APORTE AL ESTADO DEL ARTE

A la vista de los datos mostrados en este estudio, podemos concretar que la arquitectura que mejor ha rendido en términos de tiempo y de Speed-up ha sido x86 x64 de Intel. En todos los casos el rendimiento obtenido por la arquitectura Intel ha sido muy superior, como se esperaba.

A priori se consideraba que estos datos iban a ser así. Teniendo en cuenta la frecuencia por procesador la máquina Intel se prevé mejor respecto al tiempo que la ARM. Ahora bien, como se ha planteado en la sección estado del arte, la verdadera potencia de ARM radica en el consumo energético. Si resumimos quien ha obtenido los mejores resultados en cada característica medida en cada prueba obtenemos conclusiones relevantes:

Prueba	Tiempos		Speed-Up		Energía	
	ARM	INTEL	ARM	INTEL	ARM	INTEL
blackscholes						
bodytrack						
canneal						
dedup						
facesim						
ferret						
fluidanimate						
freqmine						
raytrace						
streamcluster						
swaptions						
vips						
x264						

Tabla 40 Mejor arquitectura respecto a Tiempo Speed-Up y consumo energético.

Se indica en verde que arquitectura ha obtenido mejores resultados en términos de tiempo, Speed-up y consumo energético. Las pruebas en blanco no se han podido comparar.

Como se puede ver a simple vista ARM ha supuesto una ventaja únicamente a nivel energético. Las aplicaciones que ha sido posible probar han sido claramente más eficientes a nivel energético. Este hecho pone de manifiesto que ante el mismo problema, la arquitectura ARM responde con un consumo mejor.

Por otro lado Intel ha conseguido pasar todas las pruebas sin errores. ARM en cambio ha requerido de un tiempo de compilación muy alto, y errores durante la ejecución debido a la demanda de recursos de los que no dispone. Pese a ello se conseguido el objetivo que se planteó

al principio de este documento, de conseguir portar un benchmark para máquinas de uso en sobremesa a una arquitectura pensada claramente para tecnología móvil.

En lo referente a los ámbitos de posible aplicación de la arquitectura ARM se considera que puede aportar una ventaja al tratar gran cantidad de trabajo no prioritario. Es decir, se recomienda su uso para trabajos con fuerte carga de trabajo, pero que no requieran resultados inmediatos. Por otro lado se desaconseja su uso en entornos complejos de simulación en tiempo real sin una máquina más potente que la empleada en este estudio.

Desde el punto de vista de los modelos de programación, se desaconseja emplear una máquina poco potente en modelos como el pipeline, con mucha comunicación sobre todo, los tiempos de espera afectan increíblemente al Speed-up conseguido. Extraídos los datos, estamos en disposición de contestar a las preguntas planteadas en la Introducción de este documento.

¿Puede un procesador pensado para tecnología móvil, como los desarrollados por ARM competir con un procesador "convencional" en rendimiento y ahorro energético?

En general este estudio ha demostrado que la arquitectura ARM supone una alternativa real en términos energéticos. En pequeñas escalas un vatio no puede significar mucho, pero en un clúster por ejemplo, un ahorro de 1 vatio por nodo puede suponer ahorrar miles de Euros aparte de reducir el impacto en el medio ambiente. Si ARM consigue mantener los bajos niveles de consumo energético aumentando el rendimiento, puede poner en serios aprietos a competidores como Intel.

¿Existe alguna suite de benchmarking que pueda ser usada para comparar el rendimiento de los procesadores ARM y de los procesadores de uso comercial?

Parsec 2.1 ha demostrado ser una suite completa, portable y muy adaptable en ARM, el entorno de pruebas, las herramientas para su gestión y ejecución han permitido en gran medida automatizar el proceso de recogida de datos.

En referencia al paralelismo, Parsec para ARM se ha comportado realmente bien con los diferentes APIs de programación paralela. Siendo únicamente limitado por las capacidades de la máquina. Todas las bibliotecas han podido funcionar en ARM con ligeras adaptaciones como Intel TBB.

Este estudio aporta por tanto un marco de referencia para futuras comparaciones de ambas arquitecturas. Además ofrece un *benchmark* portado a la arquitectura usable por diferentes tipos de máquinas ARM.

Abre vías a posibles estudios de uso de la arquitectura ARM en supercomputación, donde los recursos para la investigación son cada vez más escasos y el ahorro es realmente importante. Arroja luz sobre la incógnita de si una tecnología pensada para el mercado móvil tiene tendencia a sustituir al sobremesa.

3.5.1 CONCLUSIÓN FINAL

A la luz de los datos experimentales nos aventuramos a decir que la arquitectura ARM es útil en entornos de trabajo en los cuales el ahorro energético sea un factor relevante, así como el ahorro económico. Además confirmamos que la arquitectura es óptima para dispositivos con batería, como los teléfonos móviles, en los cuales la energía consumida es un factor crítico que limita la movilidad de los mismos. En aplicaciones centradas en tratamiento de datos, previsión de factores económicos o análisis de imágenes la arquitectura ha obtenido unos resultados aceptables.

Por contrapartida, la arquitectura Intel es idónea para trabajos que requieran un rendimiento alto. Con los resultados obtenidos podemos decir que la arquitectura ha funcionado muy bien en programas de simulación, codificación y en general en cualquier programa que requiera inmediatez.

4. PLANIFICACIÓN Y PRESUPUESTO

4.1 PLANIFICACIÓN

Se ha desarrollado este proyecto siguiendo la siguiente planificación:

- Estudio previo de la arquitectura Intel: Conocer todos los aspectos relativos al juego de instrucciones, historia, tecnologías empleadas en la actualidad. Recopilación de documentación y estudios realizados previamente.
- Estudio previo de la arquitectura ARM: Investigar los usos, el juego de instrucciones, disponibilidad de software, recopilación de documentación y estudios realizados previamente.
- Estudio de Parsec 2.1: Instalación, uso, portabilidad, detalle de las pruebas, relevancia en la comunidad científica, recopilación de documentación y estudios previos.
- Instalación de Parsec en Intel: Parcheado, Compilación y pruebas de test básicos de funcionalidad.
- Pruebas en Intel: Preparación de la batería de pruebas y ejecución de las mismas.
- Instalación de Parsec en ARM: Compilación y pruebas de test básicos de funcionalidad.
- Pruebas en ARM: Preparación de la batería de pruebas y ejecución de las mismas.
- Análisis de resultados: Comprobar que los datos obtenidos han sido correctos, analizar la validez de los mismos.
- Redacción de la Memoria: Unificación de los resultados de las pruebas en un documento, escritura del documento en sí.

Nombre de tarea	Duración	Comienzo	Fin
Estudio Comparativo de las capacidades de Intel y ARM	**118 días**	**lun 17/02/14**	**mié 30/07/14**
1 Estudio previo de la arquitectura Intel	10 días	lun 17/02/14	vie 28/02/14
2 Estudio previo de la arquitectura ARM	10 días	lun 24/02/14	vie 07/03/14
3 Estudio de Parsec 2.1	15 días	mar 04/03/14	lun 24/03/14
4 Instalación de Parsec en Intel	11 días	lun 31/03/14	lun 14/04/14
5 Pruebas en Intel	10 días	mar 15/04/14	lun 28/04/14
6 Instalación de Parsec en ARM	25 días	lun 21/04/14	vie 23/05/14
7 Pruebas en ARM	30 días	lun 02/06/14	vie 11/07/14
8 Análisis de resultados	10 días	lun 07/07/14	vie 18/07/14
9 Redacción de la Memoria	23 días	lun 30/06/14	mié 30/07/14

4.2 PRESUPUESTO

Se estima que el coste del proyecto se puede desglosar en recursos materiales, humanos y costes indirectos. Dado que este proyecto no tiene fines comerciales no se ha contado con estimaciones de riesgos, costes de mantenimiento o costes de seguros.

Recursos materiales:

Tipo	***Recurso***	***Coste***
Material	INTEL CORE I5-2500	400 € + 84 € IVA
	SAMSUNG EXYNOS4412 CORTEX-A9	95 € + 19,95 € IVA
	Software Office 2010	119 € + 24,95€ IVA
	Software Libre: • Ubuntu 13 • Linaro • Perf • Parsec 2.1	Gratuito
Humano	720 horas de un Ingeniero Informático a razón de 7 € por hora trabajada	6608 €
	24 horas de apoyo del Tutor a razón de 8 € por hora trabajada	192 €
	6 horas de apoyo del Director a razón de 15 € la hora	90 €
Indirectos	20% del resto de recursos	1526,58 €

TOTAL 9159,48 €

5. TRABAJOS FUTUROS

Respecto a los trabajos derivados de este estudio, se considera oportuno continuar con esta investigación en los siguientes proyectos:

- Realizar las pruebas con un medidor de energía como *Watts up?*
  - https://www.wattsupmeters.com/secure/products.php?pn=0

- Realizar los test en máquinas más potentes en ARM y con procesadores de gama alta de Intel, con tecnología como *Hyper-Threading*. Permitiría conocer si algunos de los test que han fallado por los recursos de la máquina ARM realmente son viables en los campos que estudian.

- Realizar estudios más profundos de los test individualmente, analizando las secciones críticas, escalabilidad de los *benchmarks*, análisis profundo de la gestión de memoria.

- Tratar de estudiar la viabilidad del uso ARM en clústeres de procesadores, empleando versiones adaptadas de los *benchmarks* de Parsec que dividan la carga entre varias máquinas. En estos casos el interés por el consumo energético es realmente interesante, puesto que los en este momento se plantea realizar grandes supercomputadores empleando esta tecnología y no se tienen muchos datos al respecto. En cambio los procesadores Intel y AMD son ampliamente usados en supercomputadores, con mayor gasto energético que la tecnología de Acorn.

6. BIBLIOGRAFÍA

1. *The Free Lunch Is Over - A Fundamental Turn Toward Concurrency in Software.* **Sutter, Herb.** 23, s.l. : C/C++ Users Journal, 2005.

2. *Evolution of Processor Architecture in Mobile Phones.* **Jain, Manoj Kumar.** 4, s.l. : International Journal of Computer Applications, 2014, Vol. 90.

3. *Moore's Law: past, present and future.* **Schaller, Robert R.** s.l. : IEEE, 1997.

4. **Levy, Markus.** The history of the ARM architecture: From inception to IPO. [Online] 2005. http://reds.heig-vd.ch/share/cours/reco/documents/thehistoryofthearmarchitecture.pdf.

5. **Mick, Jason.** Daily Tech. *Despite Shipments of 10 Billion chips in 2013 ARM takes Hit on Slow Growth.* [En línea] 4 de Febrero de 2014. http://www.dailytech.com/Despite+Shipments+of+10+Billion+Chips+in+2013+ARM+Takes+Hit+on+Slow+Growth/article34261.htm.

6. **ARM.** ARM Architecture Reference Manual. [En línea] 2005. https://www.scss.tcd.ie/John.Waldron/3d1/arm_arm.pdf.

7. **Corporation, Intel.** *40 años de procesadores Intel.* [PDF] 2011.

8. —. Intel® 64 and IA-32 Architectures Software Developer's Manual V2. [En línea] 2014. http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.html.

9. **Srinivasan, Sundar.** An Engineer's Options & Futures. *Intel x86 Processors – CISC or RISC? Or both??* [En línea] 2009. http://sunnyeves.blogspot.com.es/2009/07/intel-x86-processors-cisc-or-risc-or.html.

10. **Barney, Blaise.** Introduction to Parallel Computing. [Online] https://computing.llnl.gov/tutorials/parallel_comp/.

11. **Butenhof, Davi R.** *Programming with POSIX Threads.* 2008. ISBN 0201633922.

12. **OpenMP.** OpenMP Application Program Interface. *www.openmp.org.* [En línea] 2011. http://www.openmp.org/mp-documents/OpenMP3.1.pdf.

13. **Reinders, James.** s.l. : O'reilly. ISBN-10: 0-596-51480-8.

14. *The PARSEC Benchmark Suite: Characterization and Architectural Implications.* **Bienia, Christian, y otros, y otros.** s.l. : Princeton University Technical Report, 2008.

15. **Comunidad de Desarrolladores.** Linux kernel profiling with perf. [En línea] https://perf.wiki.kernel.org/index.php/Tutorial.

16. **Intel.** Intel® Core™ i5-2500 Processor. [En línea] http://ark.intel.com/products/52209/Intel-Core-i5-2500-Processor-6M-Cache-up-to-3_70-GHz.

7. ANEXO

7.1 ESPECIFICACIONES ODROID

CPU Module	Exynos4412 Quad Core CPU module Eposy molded on base board and not detachable
LAN/USB Hub	LAN9514 4-port Hi-Speed USB 2.0 hub and 10/100 Ethernet controllers from SMSC
Audio CODEC	MAX98090 is a full-featured and high performance audio CODEC from MAXIM
DC/DC Converter	RP505K331B for 3.3Volt system power supply from RICOH
Level shifter	FXMA2102L8X for I2C/UART/HDMI voltage translator from Fairchild semiconductor
Camera (Option)	S5KECGMIPI 2-Lane camera interface on rear side from Samsung
Board to Board connector	for CPU board and Application board from Uju Electronics
HDMI	Standard Micro-HDMI, supports up to 1920 x 1080 resolution
IO Port	50pin 2,54mm pitch box-header connector Interface signals for LCD-RGB, PWM, ADC, SPI, I2C, UART and GPIO.
DC Input	5V / 2A input, Plug specification is inner diameter 0.8mm and outer diameter 2.5mm